01 Jul 1988

Complete Sets of Reductions Modulo A Class of Equational Theories which Generate Infinite Congruence Classes

Timothy B. Baird

Ralph W. Wilkerson
Missouri University of Science and Technology, ralphw@mst.edu

COMPLETE SETS OF REDUCTIONS MODULO A CLASS OF EQUATIONAL THEORIES WHICH GENERATE INFINITE CONGRUENCE CLASSES

T. B. Baird* and R. W. Wilkerson

CSc-88-5

Department of Computer Science

University of Missouri-Rolla

Rolla, Missouri 65401 (314)341-4491

*This report is substantially the Ph.D. dissertation of the first author, completed, July 1988.

ABSTRACT

In this paper we present a generalization of the Knuth-Bendix procedure for generating a complete set of reductions modulo an equational theory. Previous such completion procedures have been restricted to equational theories which generate finite congruence classes. The distinguishing feature of this work is that we are able to generate complete sets of reductions for some equational theories which generate infinite congruence classes. In particular, we are able to handle the class of equational theories which contain the associative, commutative, and identity laws for one or more operators.

We first generalize the notion of rewriting modulo an equational theory to include a special form of conditional reduction. We are able to show that this conditional rewriting relation restores the finite termination property which is often lost when rewriting in the presence of infinite congruence classes. We then develop Church-Rosser tests based on the conditional rewriting relation and set forth a completion procedure incorporating these tests. Finally, we describe a computer program which implements the theory and give the results of several experiments using the program.

Key Words: complete sets of reductions, Knuth-Bendix procedure, E-completion, E-unification, conditional reductions, finite termination property, Church-Rosser property.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Ralph Wilkerson, for his guidance and support during the course of my studies and research. I am also grateful for the assistance of the other members of my advisory committee: Dr. Arlan DeKock, Dr. Peter Ho, Dr. Ronald Kellogg, and Dr. George Zobrist. Special thanks also go to Dr. Gerald Peterson of McDonnell-Douglas in Saint Louis, Missouri, who first suggested this research topic and who has shared in every phase of this project. His insight and assistance are deeply appreciated.

I would also like to thank Dr. Dean Priest and Dr. Steve Smith of Harding University in Searcy, Arkansas, who arranged for my leave of absence to pursue this degree. I am very grateful for the financial assistance provided by Harding University these last three years.

The work presented in this document is but a part of a larger group effort in the Computer Science department at the University of Missouri-Rolla. I would like to thank all of those involved in this research effort. Fellow graduate students Blayne Mayfield and Barbara Smith have provided stimulating discussion, encouragement, assistance, friendship, and empathy throughout our work together. Sincere thanks is expressed to each of them. The program which implements the theory presented in this document is also a product of a group effort. Contributions were made by Dr. Wilkerson, Dr. Peterson, Barbara Smith, and our local LISP expert, Blayne Mayfield.

Many friends and family members have offered encouragement throughout this undertaking. I especially thank my parents for all their love. Most of all, I thank my wife, Debbie, whose love and friendship make all things worthwhile. Without her continued support and assistance, this work could not have been done.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

I. INTRODUCTION

We begin by giving a concise, though somewhat informal description of the problem addressed by this research. More formal and complete discussions of each essential element will be given in later chapters. After setting forth the specific purpose of this research we discuss why this problem is worthy of our attention. Finally, we preview the structure and content of the remainder of the chapters.

A. OBJECTIVE

Briefly stated, a complete set of reductions for a given algebraic system is defined such that any two terms which are congruent under the axioms of the algebraic system must have identical forms after the set of reductions has been applied exhaustively to each. Whenever a complete set of reductions can be found for a given algebraic system it eliminates the unmanageable search space often encountered in equational theorem proving, providing a very efficient tool for solving equality problems relative to the axioms of the system.

Knuth and Bendix [*KB*70] first established two necessary and sufficient conditions for a set of reductions to be complete. These conditions have come to be called the finite termination property and the confluence property. Based on these conditions they were able to devise both an algorithm for testing the completeness of a set of reductions and a procedure which can take the equational axioms of an algebraic system and possibly generate a complete set of reductions. We will refer to their procedure as the Knuth-Bendix completion procedure and to all similar procedures as completion procedures. The Knuth-Bendix procedure was able to generate complete sets of reductions for a limited number of algebraic systems, most notably free groups. Early completion procedures, however, were not able to handle any algebraic system whose definition included a commutativity axiom because

inclusion of these axioms in the reduction set resulted in the loss of the finite termination property.

Peterson and Stickel [*PS*81] were able to overcome this limitation of completion procedures by splitting the equational axioms of an algebraic system into two sets: (1) equations which are incorporated into the pattern matching process used to apply reductions, and (2) equations which form the basis of a set of reductions to be completed. Their approach requires not only the finite termination and confluence properties, but also a linearity property for equations in the first set and a special compatibility property between the reductions and the first set of equations. Besides these properties it is necessary to have a finite and complete unification algorithm for the equations which are incorporated into the pattern matching process. Peterson and Stickel were able to generate complete sets of reductions for algebraic systems which included both associativity and commutativity axioms, building these axioms into the pattern matching facility via associative/commutative unification. Such completion procedures have come to be called E-completion procedures, where E represents the set of equations incorporated into the pattern matching process. Using this E-completion procedure, Peterson and Stickel were able to generate complete sets of reductions for algebraic systems such as commutative groups, commutative rings, and distributive lattices.

Jouannaud and Kirchner [*JK*86] generalized the theory of E-completion sufficiently to account for all previous completion and E-completion theory. They were able to replace the compatibility requirement of Peterson and Stickel with a more general property which they call coherence and to remove the linearity requirement for E in favor of the more general requirement that the congruence classes generated by E must be finite. In the introduction to their paper, Jouannaud and Kirchner state:

> Our proof holds for the particular case of Peterson and Stickel's rewriting relation, without any linearity hypothesis on rules or equations. However, the case of infinite congruence classes remains the last open problem of the theory of equational term rewriting systems.

They also return to this point in their conclusion, stating:

> ... the last open problem of infinite congruence classes should be addressed, since many interesting cases such as equipotency and identity fall in this category.

It is the goal of this research to attack the problem of E-completion when the set E generates infinite congruence classes. Rather than attempting to solve this problem for all equational theories which generate infinite congruence classes, however, we address only the class of equational theories which contain the associative, commutative, and identity laws for one or more operators. We will call these ACI equational theories and the corresponding E-completion process we will call ACI-completion. It is the presence of the identity law in these equational theories which causes them to generate infinite congruence classes and thus fall outside the realm of previous E-completion theory. In the chapters that follow we will develop, implement, and experiment with a new theory for E-completion which handles the class of ACI equational theories.

Others have addressed the problem of infinite E-congruence classes. Bachmair and Plaisted [*BP*87] have generalized the theory of Jouannaud and Kirchner so as to have apparently removed the finite congruence class requirement. They do, however, still maintain the other requirements of Jouannaud and Kirchner, including the finite termination property. We will demonstrate that the finite termination property needed in their model is usually lost when the equational theory generates infinite congruence classes, leaving the real issue of an implementable E-completion procedure in the presence of infinite E-congruence classes as an open problem.

B. MOTIVATION

Why do we want to develop an E-completion procedure for ACI equational theories? Not only is this an interesting and challenging open problem, we see three benefits which may be realized from the solution: (1) increased step size for equality inferences, (2) increased understanding of the essential elements of E-completion procedures, and (3) a pattern matching process which is more closely akin to the process used by a human mathematician. We now discuss each of these benefits in turn.

It is generally recognized that one of the major problems in the area of automated reasoning is the development of inference rules which take deduction steps of the appropriate size [*Wo*88]. The resolution principle [*Ro*65], though theoretically complete, suffers greatly when dealing with equality because of a step size which is too small. Ideally, we would like to increase step size without sacrificing completeness. Demodulation [*Wo*67], paramodulation [*WR*69], and complete sets of reductions have all been developed to address this problem.

From the development of previous E-completion procedures it is easy to observe that the step size of an equality inference becomes larger whenever the congruence classes generated by E become larger. When the set E is empty each inference step clashes two individual clauses to produce an individual clause. When E generates congruence classes, however, each inference step clashes all of the clauses in one congruence class with all of the clauses in another congruence class in a single operation, producing a resultant clause which stands by itself in place of all of the clauses in its congruence class. Consequently, we find fewer inferences are needed to cover the same ground and fewer reductions are needed to constitute a complete set of reductions for a given algebraic system.

Because each inference is accomplishing more by itself and fewer reductions are needed, the branching factor of the search tree and the resulting size of the search space are both reduced. If we are able to develop pattern matching facilities for the larger equational theories which are as efficient as those for smaller theories, we may be able to generate complete sets of reductions for algebraic systems which have been unattainable by previous E-completion systems. Furthermore, a generalized E-completion theory which handles infinite E-congruence classes may allow us to attempt problems which were not feasible under previous E-completion theory. An ACI-completion procedure is a small, first step in that direction.

Besides the possibility that increased step size of equality inferences may lead to the solution of new problems, there is the benefit that the development of an ACI-completion procedure will lead to a better understanding of E-completion procedures in general. Generalization of a theory necessitates that essential and inessential elements are more clearly distinguished. The concepts presented here, or others that spring from them, may eventually lead to improving the performance of automated reasoning systems which deal with equality inferences. More specifically, we believe that the solution of the E-completion problem for this one class of equational theories which generates infinite congruence classes provides insight into how to attack the larger problem of E-completion for all equational theories which generate infinite congruence classes.

As a third benefit, we point out that incorporating associativity, commutativity, and identity laws into the pattern matching process seems intuitively to be more like the way human mathematicians deal with these axioms. In particular, the identity law, despite its inherent simplicity, is a source of great difficulty for all previous E-completion theory. Yet this law does not seem to cause any real difficulty for the experienced mathematician. The core element and conditional reduction approach

which we develop in this research is inspired by the way we believe we approach these problems when we are working without the aid of the computer.

C. STRUCTURE

It is our intention that this document be sufficiently self-contained so that a reader unfamiliar with the area of term rewriting systems will be able to find the necessary background material here. Of course, the references cited may be used to fill in any gaps. We do assume that the reader is generally familiar with systems of mathematical notation.

In Chapters 2 and 3 we give background information which is needed to understand the theory which is developed in later chapters. Chapter 2 gives basic vocabulary and definitions associated with completion theory, as well as a brief review of pattern matching algorithms which are essential ingredients of completion procedures. Chapter 3 is a detailed literature review of the theory of complete sets of reductions and completion procedures.

In Chapters 4 and 5 we develop our theory of ACI-completion. Chapter 4 presents a new type of conditional rewriting relation as a method for establishing the finite termination property in the presence of infinite ACI-congruence classes. In Chapter 5 we first develop tests for completeness modulo an ACI equational theory based on the new conditional rewriting relation and then give an ACI-completion procedure.

In Chapters 6 and 7 we report results of implementing the given theory in a computer program. Chapter 6 describes several experiments which were performed and Chapter 7 presents conclusions as well as ideas for further research.

II. PRELIMINARIES

Before we can present the necessary mathematical theory relative to complete sets of reductions and completion procedures we must review the definitions and concepts on which we will build. In this chapter we first give definitions for those concepts which are needed as general background for all subsequent chapters. More specific concepts will be defined and discussed in later chapters as they are needed. After presenting these background definitions, we present an overview of unification algorithms. Although unification is not the primary focus of this research, it is extremely important because of its key role in completion procedures.

A. DEFINITIONS

We have grouped the background definitions into four major categories: (1) definitions related to terms, (2) definitions related to unifiers, (3) definitions related to equational theories, and (4) definitions related to rewriting relations. Much of the material in this section is adapted from [*PS*81] and [*JK*86].

1. Terms.

We assume the existence of a countably infinite set, V, of *variables* and a finite set, F, of *operators* such that $V \cap F = \phi$. With each operator we associate a *degree* which indicates the number of operands on which it operates. Operators of degree zero are called *constants*. Constants and variables are called *simple terms*. *Complex terms* are formed when an operator f of degree n is paired with an ordered n-tuple of operands, each of which may be simple or complex terms. Complex terms may be written in *prefix form* as $f(x,y)$ or in *infix form* as $x f y$. The set $T(F,V)$ represents the set of all possible terms which may be formed using elements of F and V, consistent with the degree of each operator.

We use a standard *tree representation* for terms as follows: simple terms are represented by leaf nodes; a complex term t which is formed from an operator f of degree n is represented by an n-ary tree where the root node represents f and the n children from left to right represent the ordered operands of f. For example, if x and y are variables, 0 is a constant, $+$ and $*$ are binary operators, and $-$ is a unary operator, then the term $t = (x + (0*(-y)))$ is represented by the *term tree* shown in Figure 1.

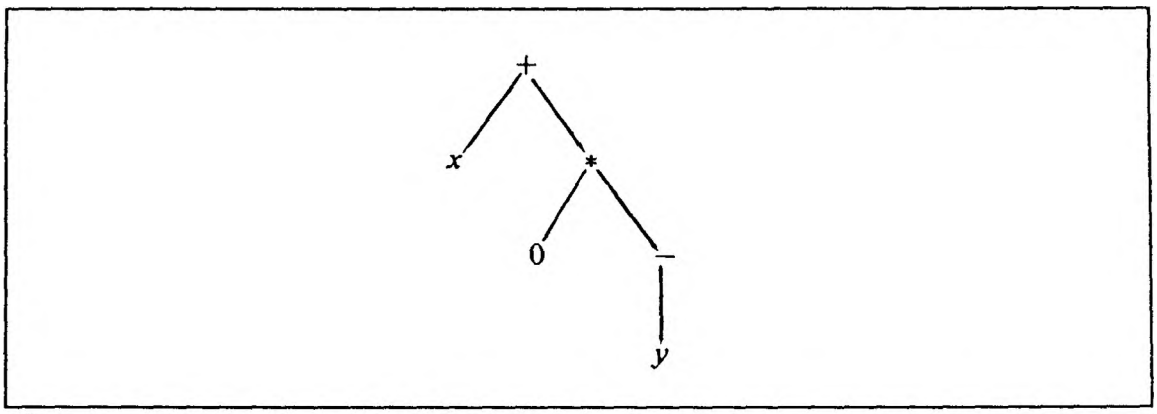

Figure 1. Tree representation for a term

Let *Parent*(*node*) represent the parent of a node and *Cpos*(*node*) be the position of a child relative to its sibling nodes. We define a *position* function for the nodes of a term tree by $Pos(root) = \varepsilon$ and $Pos(node) = Pos(Parent(node)).Cpos(node)$. For convenience we will write positions such as $\varepsilon.m$ as simply m. In the example above $Pos(+) = \varepsilon$, $Cpos(x) = 1$, $Cpos(*) = 2$, $Pos(x) = 1$, $Pos(*) = 2$, $Pos(0) = 2.1$, $Pos(-) = 2.2$, and $Pos(y) = 2.2.1$.

A *subterm* of a term is the term associated with a subtree of a term tree. We write t/m to indicate the subterm of t at position m. In our example $t/2.2 = (-y)$. The subterm t/m is said to be a *strict subterm* whenever $m \neq \varepsilon$. We use the notation $t[m \leftarrow s]$ to indicate the term which is obtained when we replace the subterm t/m by

another term s, without altering the remainder of t. In our running example, $t[2.2 \leftarrow a] = x + (0 * a)$. We define the *root operator* of a complex term t, $t.root$, to be the operator at the root of the term tree for t.

We define $Vars(t)$ to be the set of all variables occurring in t. For our example, $Vars(t) = \{x, y\}$. A term t is defined to be *linear* iff for all $x \in Vars(t)$, x occurs exactly once in t. Terms t_1 and t_2 are said to be *variable disjoint* whenever $Vars(t_1) \cap Vars(t_2) = \phi$.

We define the *domain* of a term t, $Dom(t)$, as the set of all positions occurring in the term tree for t. For our example, $Dom(t) = \{\varepsilon, 1, 2, 2.1, 2.2, 2.2.1\}$. The *first level domain* of a term t, $Fdom(t)$, is defined to be the set of positions for all nodes which are immediate children of $t.root$. $Fdom(t) = \{1, 2\}$, for our example. We also define the *strict domain* of a term t, $Sdom(t)$, by

$$Sdom(t) = \{p \mid p \in Dom(t) \text{ and } t/p \notin Vars(t)\}.$$

$Sdom(t)$ represents the positions of all of the subterms of t which are not variables. These are sometimes referred to as the positions of the *non-trivial subterms* of t. For our example, $Sdom(t) = \{\varepsilon, 2, 2.1, 2.2\}$.

For terms containing an operator $+$ which satisfies the associative law $(x + y) + z = x + (y + z)$, we may sometimes use a simplifying representation. We will say that a term t has been *flattened* whenever all subterms rooted with the same associative operator as their parent term have been collapsed, removing the operator of the subterm and placing the operands of the subterm in the scope of the parent term's operator. For example, the terms $(a + b) + (c + d)$ and $(((a + b) + c) + d)$ both have the flattened representation $a + b + c + d$. This representation requires that we no longer view $+$ as an operator of degree two, but as an operator of arbitrary degree.

2. Unifiers.

We define a *substitution pair* to be an ordered pair (v,t) and is usually written as $v \leftarrow t$, where v is a variable and t is a term. A *substitution* is then defined to be a set of substitution pairs $\{v_1 \leftarrow t_1,\ v_2 \leftarrow t_2,\ \ldots,\ v_n \leftarrow t_n\}$ such that each v_i occurs exactly once in the n-tuple $(v_1, v_2, \ldots, v_n)$. We say that a substitution σ is *applied* to a term t, usually written as simply $t\sigma$, whenever for every $v_i \leftarrow t_i \in \sigma$ we simultaneously replace all occurrences of v_i in t with t_i. For example, consider the substitution $\sigma = \{x \leftarrow a,\ y \leftarrow b + x\}$ and the term $t = (x + y) - x$. Applying σ to t gives $t\sigma = (a + (b + x)) - a$.

Whenever we have two substitutions, σ_1 and σ_2, we obtain their *composition*, $\sigma_1\sigma_2$, by the following:

$$\sigma_1\sigma_2 = \{v_i \leftarrow t_i\sigma_2 \mid v_i \leftarrow t_i \in \sigma_1\} \cup \{v_i \leftarrow t_i \mid (v_i \leftarrow t_i \in \sigma_2 \text{ and } \forall(v_j \leftarrow t_j) \in \sigma_1\ v_i \neq v_j)\}$$

The desired consequence of this definition is that $(t\sigma_1)\sigma_2 = t(\sigma_1\sigma_2)$. In other words, we get the same result from composing two substitutions and then applying the composed substitution as we do from applying the individual substitutions one after the other. Whenever no variable occurring in either side of a substitution pair of σ_1 occurs in either side of a substitution pair of σ_2 we say that σ_1 and σ_2 are *variable disjoint*. For variable disjoint substitutions $\sigma_1\sigma_2 = \sigma_2\sigma_1$, and thus, order of composition has no effect.

Whenever a substitution σ satisfies the equation $t_1 = t_2\sigma$ we say that σ is a *matching substitution* or simply a *matcher* for terms t_1 and t_2. We also say that t_1 is an *instance* of t_2. Suppose that $t_1 = x + 0$ and $t_2 = a + 0$, then $\sigma = \{x \leftarrow a\}$ is a matcher for t_1 and t_2. Whenever a substitution σ satisfies the equation $t_1\sigma = t_2\sigma$ we say that σ is a *unifying substitution* or simply a *unifier* for t_1 and t_2. For example, when $t_1 = x + 0$ and $t_2 = a + y$ $\sigma = \{x \leftarrow a,\ y \leftarrow 0\}$ is a unifier for t_1 and t_2. For variable disjoint

terms it should be clear that a matcher is always a unifier, but a unifier is not always a matcher. Matchers and unifiers are sometimes referred to as *one-way unifiers* and *two-way unifiers*, respectively.

We now point out that unifiers are not necessarily unique. For example, $\sigma_1 = \{u \leftarrow x + a,\ v \leftarrow y + b\}$ and $\sigma_2 = \{u \leftarrow z + a,\ x \leftarrow z,\ v \leftarrow w + b,\ y \leftarrow w\}$ are both unifiers for $t_1 = (x + a) + (y + b)$ and $t_2 = u + v$ Clearly there are an infinite number of unifiers which follow the form of σ_2, as z and w can be replaced by any valid terms and the result is still a unifier. We say that a substitution σ_2 is an *instance* of another substitution σ_1 iff there exists a third substitution σ_3 such that $\sigma_2 = \sigma_1\sigma_3$. Note that in our last example σ_2 is an instance of σ_1. This is easily seen using $\sigma_3 = \{x \leftarrow z,\ y \leftarrow w\}$. We say that σ_1 is *more general* than σ_2 whenever σ_2 is an instance of σ_1. We define a substitution θ to be a *most general unifier* for terms t_1 and t_2 whenever all other unifiers of t_1 and t_2 are instances of θ.

A *variable renaming substitution* is a substitution of the form $\{x_1 \leftarrow y_1,\ x_2 \leftarrow y_2,\ \ldots,\ x_n \leftarrow y_n\}$ where all of the x_is are disjoint variable names and all of the y_is are also disjoint variable names. Two substitutions, σ_1 and σ_2, are said to be *the same modulo variable renaming* whenever there exists variable renaming substitutions θ_1 and θ_2 such that $\sigma_2 = \sigma_1\theta_1$ and $\sigma_1 = \sigma_2\theta_2$. Robinson proved in [*Ro65*] that a most general unifier, when it exists, is unique modulo variable renaming.

The process of finding matchers is called *matching*. The process of finding unifiers is called *unification*. When the terms being unified are variable disjoint, as will always be the case in the completion theory which follows, it is easy to see that a procedure which performs unification can be used to perform matching. If we treat all of the variables in one of the terms as constants then the unifier generated by unification is also a matcher. Because of this close conceptual connection between

the two processes we will focus our discussion of matching and unification algorithms on the problems of unification.

3. Equational Theories.

An *equation* is an ordered pair of terms (l,r), usually written as $l = r$. An *equational theory*, E, is a set of equations. If E is an equational theory, then $E^c = \{r = l \mid l = r \in E\}$. For an equational theory E we define the *one step E-equality* relation, $\overset{E}{\underset{1}{=}}$, on pairs of terms as follows:

$s \overset{E}{\underset{1}{=}} t$ iff there exists (1) an equation $l = r \in E \cup E^c$,

(2) a node $n \in Dom(s)$, and

(3) a substitution θ

such that $s/n = l\theta$ and $t = s[n \leftarrow r\theta]$.

The *E-equality* relation, $\overset{E}{=}$, is then defined to be the reflexive, symmetric, transitive closure of $\overset{E}{\underset{1}{=}}$, which is clearly an equivalence relation. Whenever an E-equality relation satisfies the property $s \overset{E}{=} t \Rightarrow f[m \leftarrow s] \overset{E}{=} f[m \leftarrow t]$ for all terms f, s, and $t \in T(V,F)$, then the E-equivalence relation is said to be *compatible* with the term structure for $T(V,F)$. The *E-equivalence class* for a term t, $[t]_E$, is defined by $[t]_E = \{s \mid s \in T(V,F) \text{ and } s \overset{E}{=} t\}$. When $\overset{E}{=}$ is an equivalence relation which is also compatible with the term structure for a set of terms, then the E-equivalence class is said to be an *E-congruence class*. Suppose that we have $E = \{x + y = y + x\}$. We can show that $a + b \overset{E}{=} b + a$ using $x + y = y + x$ as $l = r$, ε as n, and $\{x \leftarrow a, y \leftarrow b\}$ as σ. Similarly, we can show that

$$[a + (b + c)]_E = \{a + (b + c),\ a + (c + b),\ (b + c) + a,\ (c + b) + a\}.$$

E-membership for sets, $\in_E$, is defined by $s \in_E S$ iff there exists a s' such that $s \overset{E}{=} s'$ and $s' \in S$. Thus, using E from the previous example, we would say that

$b + a \in_E \{a, b, a + b\}$. Finally, we define *E-equality for substitutions*, σ and σ', by $\sigma \stackrel{E}{=} \sigma'$ iff $p \in \sigma$ implies $p \in_E \sigma'$ and $p' \in \sigma'$ implies $p' \in_E \sigma$.

The following lemma states that E-equality is preserved under the application of substitutions, or equivalently, that E-equality is compatible with the application of substitutions:

Compatibility Lemma [*PS*81]: Suppose $s, t \in T(V,F)$, θ is a substitution, and E is an equational theory. If $s \stackrel{E}{=} t$, then $s\theta \stackrel{E}{=} t\theta$.

The problems of matching and unification are easily redefined in the presence of equational theories. We say that a substitution is an *E-matcher* for terms t_1 and t_2 whenever σ satisfies the equation $t_1 \stackrel{E}{=} t_2\sigma$. The process of finding such substitutions we call *E-matching*. Likewise, we say that σ is an *E-unifier* for t_1 and t_2 whenever it satisfies the equation $t_1\sigma \stackrel{E}{=} t_2\sigma$. The process of finding E-unifiers is called *E-unification*. Whenever $t_1 \stackrel{E}{=} t_2\sigma$ we also say that t_1 is an *E-instance* of t_2. A substitution σ_1 is said to be an E-instance of another substitution σ_2 whenever there exists a third substitution σ_3 such that $\sigma_1 \stackrel{E}{=} \sigma_2\sigma_3$. We indicate by $\iota_E\Theta$ the set of all substitutions which are E-instances of substitutions in the substitution set, Θ.

The problem of E-unification has been studied for many different equational theories. The existence of a most general E-unifier which is unique modulo variable renaming does not generally hold for E-unification. For example, when we have the equational theory $E = \{x + y = y + x\}$ it is easy to see that both $\sigma_1 = \{x \leftarrow c + d, y \leftarrow a, z \leftarrow b\}$ and $\sigma_2 = \{x \leftarrow a + b, y \leftarrow c, z \leftarrow d\}$ are E-unifiers for terms $t_1 = x + (y + z)$ and $t_2 = (a + b) + (c + d)$, yet neither σ_1 nor σ_2 are variable renaming E-instances of the other. When E contains only the associative law $(x + y) + z = x + (y + z)$ it has been shown that there may be an infinite set of such unique E-unifiers for two terms [*St*81]. Because of these complications the following

general properties are defined for E-unification algorithms which produce a set Σ of E-unifiers for terms t_1 and t_2: [*Si*79]

finiteness: $|\Sigma| \leq \infty$.

completeness: $\forall\theta$ such that $t_1\theta \stackrel{E}{=} t_2\theta$ ($\exists\sigma \in \Sigma$ and a substitution τ such that $\theta \stackrel{E}{=} \sigma\tau$).
(All possible E-unifiers are E-instances of some E-unifier in Σ).

minimality: For no $\sigma_i, \sigma_j \in \Sigma$ is $\sigma_i = \sigma_j\theta$ where $\theta \neq \{\}$.
(No E-unifier in Σ is an E-instance of another E-unifier in Σ).

The counterpart to a most general unifier for E-unification is the existence of a finite, complete, and minimal set of E-unifiers. Finiteness and completeness are the most important properties since we can never finish producing E-unifiers without finiteness and we cannot be sure that we have found general forms for all possible E-unifiers without completeness. Minimality can always be obtained from finiteness and completeness by post-processing the set of E-unifiers and throwing out those which are E-instances of others. This post-processing is very costly, thus an algorithm which avoids producing redundant unifiers is often important for reasons of efficiency. A summary of known finite, complete, and minimal E-unification algorithms is given in [*Si*79].

A system consisting of a set, one or more *n*-ary operations on the set, and one or more relations on the set is defined to be an *algebraic structure* [*TM*75]. When all of the relations on the set can be defined by equations we will call this an *equational algebraic structure*. In this research we will be particularly interested in equational algebraic structures defined by the set $T(V,F)$ and an equational theory A where there exists another equational theory E such that $E \subseteq A$ and a finite and complete E-unification algorithm exists for E. We will focus particularly on structures where E contains associative, commutative, and identity laws (*ACI equational theories*), associative and commutative laws (*AC equational theories*), commutative laws (*C equational theories*), and where E contains no laws (*empty equational theories*).

4. Rewriting Relations.

A *rewrite rule* is an ordered pair (λ, ρ), usually written $\lambda \rightarrow \rho$, which may be *applied* to an arbitrary term t as follows: if there exists a position $m \in Dom(t)$ and a substitution σ such that $t/m = \lambda\sigma$, then the resulting term, t', is given by $t[m \leftarrow \rho\sigma]$. We say that t *rewrites* to t'. The idea is that λ and ρ are equivalent and we have substituted one for the other. Note that σ is a matcher for λ and t/m and may be found via any matching algorithm. The application of a rewrite rule is sometimes called a *substitution rule of inference.* This is the same process we use when we draw the conclusion "Mary is sick today" from the statements "John's wife is sick today" and "Mary is John's wife".

A *reduction* is a special rewrite rule where it is understood that ρ is in some sense simpler than λ. When t rewrites to t' via a reduction we say that t *reduces* to t'. Reductions are precisely the same as the *demodulators* introduced in the demodulation process of Wos [*Wo*67]. When a reduction is applied to a term t, the new version of t is equivalent to and yet simpler than the original t. For example, when we apply the reduction $e*x \rightarrow x$ to the term $t = a*(e*b)$ using $m = 2$ and $\sigma = \{x \leftarrow b\}$ the resulting term $t' = a*b$ is simpler than the original t. When no reduction in a reduction set R can be used to reduce a term t we say that t is *irreducible by R.* An irreducible term is sometimes called a *normal form* or *terminal form* of the term from which it has been derived. We use the notation $t{\downarrow}^R$ to indicate a term which has been derived from t by zero or more applications of reductions from R and is now a terminal form relative to R.

A *conditional reduction* is a reduction of the form *If C Then* $\lambda \rightarrow \rho$. The condition, C, normally involves the same variables and operators as $\lambda \rightarrow \rho$ and is evaluated after the matching substitution is found. If the condition holds the reduction is applied as usual, otherwise the process is aborted. For example, the

reduction *If* $x \neq 0$ *Then* $x*x^{-1} \rightarrow 1$ can be used to prevent rewriting with a substitution which sets x to zero, thus preventing division by zero. The most commonly studied conditional reductions are reductions where C is of the form $t_1 = u_1$ *and* $t_2 = u_2$ *and ... and* $t_n = u_n$ [*BK*86, *KR*87]. In most cases the presence of the condition on the reduction arises out of the semantics of the problem, just as in our example the semantics of the division process clearly demands that we not divide by zero. In this research we will introduce a new type of conditional reduction where the conditions are of a slightly different form and arise more from the syntax of the problem, rather than the semantics.

A set of rewrite rules, R, can be used to define a binary relation on the set of terms. The *rewriting relation* R, written as $\xrightarrow{R}$, is the set of ordered pairs (t_1, t_2) such that t_1 rewrites to t_2 using some rewrite rule from R. We write $t_1 \xrightarrow{R} t_2$ to indicate $(t_1, t_2) \in \xrightarrow{R}$. We use the notation $\xrightarrow{R}_*$ to indicate the reflexive, transitive closure of $\xrightarrow{R}$. Thus $t_1 \xrightarrow{R}_* t_2$ means that we can move from t_1 to t_2 using zero or more applications of $\xrightarrow{R}$. We also define *E-rewriting relations,* often called *rewriting modulo E,* by altering the rewriting definitions to account for an equational theory, E. This generally can be viewed as rewriting between two different E-congruence classes.

Three rewriting relations which have been used extensively in the study of completion procedures and which will be used in this research are $\xrightarrow{R}$, $\xrightarrow{R,E}$, and $\xrightarrow{R/E}$. These are defined as follows:

$$t_1 \underset{\lambda, \rho, \sigma, m}{\xrightarrow{R}} t_2 \text{ iff } \lambda \rightarrow \rho \in R,\ m \in Dom(t_1),\ t_1/m = \lambda\sigma, \text{ and } t_2 = [m \leftarrow \rho\sigma]$$

$$t_1 \underset{\lambda, \rho, \sigma, m}{\xrightarrow{R,E}} t_2 \text{ iff } \lambda \rightarrow \rho \in R,\ m \in Dom(t_1),\ t_1/m \overset{E}{=} \lambda\sigma, \text{ and } t_2 = [m \leftarrow \rho\sigma]$$

$$t_1 \underset{\lambda, \rho, \sigma}{\xrightarrow{R/E}} t_2 \text{ iff } \exists\, t'_1, t'_2 \text{ such that } t_1 \overset{E}{=} t'_1 \underset{\lambda, \rho, \sigma}{\xrightarrow{R}} t'_2 \overset{E}{=} t_2$$

The first relation, $\xrightarrow{R}$, is the standard rewriting relation which is used when the equational theory is empty. The second relation, $\xrightarrow{R,E}$, is a limited form of rewriting modulo E which is easily implemented via E-matching. If we use the notation $\underset{\leq m}{\overset{E}{=}}$ to indicate an E-equality relation where each $\underset{1}{\overset{E}{=}}$ step takes place at or below m, then an equivalent alternate definition for $\xrightarrow{R,E}$ is

$$t_1 \underset{\lambda,\rho,\sigma,m}{\xrightarrow{R,E}} t_2 \text{ iff } \lambda \to \rho \in R,\ m \in Dom(t_1),\ \text{and } \exists\, t'_1 \text{ such that } t_1 \underset{\leq m}{\overset{E}{=}} t'_1 \underset{\lambda,\rho,\sigma,m}{\xrightarrow{R}} t_2$$

The third relation, $\xrightarrow{R/E}$, is the most general form of rewriting modulo E and is used more in proofs of the theory than in implementations. The difference between $\xrightarrow{R,E}$ and $\xrightarrow{R/E}$ is sometimes very subtle. Suppose that R contains the single reduction $(-x)+x \to 0$ and E is an AC equational theory. Then the term $t_1 = (a+b)+(-b)$ can be rewritten via $\xrightarrow{R/E}$ using the sequence $t_1 \overset{AC}{=} a+((-b)+b) \xrightarrow{R} a+0$. The term t_1 cannot be rewritten at all via $\xrightarrow{R,E}$ because there does not exist a position $m \in Dom(t_1)$ and a substitution σ such that $t_1/m \overset{AC}{=} ((-x)+x)\sigma$. It should be clear from the definitions that $\xrightarrow{R} \subseteq \xrightarrow{R,E} \subseteq \xrightarrow{R/E}$.

B. UNIFICATION ALGORITHMS

Because the concepts of matching, E-matching, unification, and E-unification all play a central role in the development of completion procedures we present a brief summary of matching and unification algorithms in this section. We will focus on unification and E-unification since the matching problems are simpler instances of these. For our purposes, we are interested in unification relative to empty, C, AC, and ACI equational theories. We must also consider terms which involve various operators, each of which may be associated with a different one of these equational theories.

1. Standard Unification.

We will refer to unification relative to the empty equational theory as *standard unification* or *S-unification*. This type of unification has also been called *Robinson unification*, after its founder, and *Null-E unification* [*Ma*88]. A variation on Robinson's algorithm is given in Figure 2. Robinson [*Ro*65] proved that the standard unification algorithm will always terminate, returning the most general unifier if it exists, and failure otherwise.

Procedure **S-Unify** (t_1, t_2, Σ)

Case

1: $t_1 \stackrel{E}{=} t_2 \Rightarrow$ Return Σ

2: *Is-Var*$(t_1) \Rightarrow$ If t_1 occurs in t_2
Then Return ϕ
Else Return $\cup_{\sigma \in \Sigma} \sigma \circ \{t_1 \leftarrow t_2\}$

3: *Is-Var*$(t_2) \Rightarrow$ Return *S-Unify*(t_2, t_1, Σ)

4: *Complex*(t_1) and *Complex*(t_2) and *Length*(t_1) = *Length*(t_2) and t_1.*root* = t_2.*root* $\Rightarrow$ For each $m \in$ *Fdom*(t_1) do
$\Sigma := \cup_{\sigma \in \Sigma}$ *S-Unify*$((t_1/m)\sigma, (t_2/m)\sigma, \{\sigma\})$
End For
Return Σ

5: Otherwise $\Rightarrow$ Return ϕ

End Procedure S-Unify

Figure 2. Standard Unification Algorithm

The following examples illustrate the algorithm:

Example 1: Suppose we want to unify the terms $P(a, x, f(x))$ and $P(y, b, z)$. All of the unification algorithms which we present make use of an initial unifier set, Σ, with

which the final unifier set must be consistent. This is actually only needed for the recursive calls. All of the presented unification algorithms also return their result in the form of a set of unifiers, even though for standard unification this set can contain at most one element. The unifier set $\{\phi\}$ represents the single unifier with no substitution pairs. We will begin all top level calls using this value for Σ. The unifier set ϕ will be returned when no unifier is possible. We begin this example with the call *S-Unify*($P(a, x, f(x)), P(y, b, z), \{\phi\}$). This call is handled by Case 4 which generates the following recursive calls:

(1) *S-Unify*($a, y, \{\phi\}$): Case 3 generates *S-Unify*($y, a, \{\phi\}$) which returns $\{\{y \leftarrow a\}\}$.

(2) *S-Unify*($x, b, \{y \leftarrow a\}$): Case 2 returns $\{\{y \leftarrow a, x \leftarrow b\}\}$.

(3) *S-Unify*($f(b), z, \{y \leftarrow a, x \leftarrow b\}$): Case 3 generates *S-Unify*($z, f(b), \{y \leftarrow a, x \leftarrow b\}$) which returns $\{\{y \leftarrow a, x \leftarrow b, z \leftarrow f(b)\}\}$ via Case 2.

Example 2: Suppose we want to unify x and $f(x)$. We begin with the call *S-Unify*($x, f(x), \{\phi\}$). This is handled by Case 2, where t_1 is found to be a variable occurring in t_2. Thus the unification fails and returns ϕ.

2. Commutative Unification.

The commutative unification, or *C-unification*, algorithm given here is due to Siekmann [*Si79*]. Let *C-Permute*(t) be the set of all possible terms which may be formed by permuting the operands of all of the commutative operators in t. For example, let f be a commutative operator and let $t_1 = f(a,f(b,c))$. This gives *C-Permute*$(t_1) = \{f(a,f(b,c)), f(a,f(c,b)), f(f(b,c),a), f(f(c,b),a)\}$. Siekmann pointed out that an obvious solution to the C-unification problem for terms t_1 and t_2 is to perform S-unification for all possible pairs from *C-Permute*$(t_1) \times$ *C-Permute*(t_2). Siekmann showed that this approach is finite and complete, though not minimal. In fact, this approach is very inefficient.

This led Siekmann to modify the obvious solution as follows: Let *C-Oprs*(t) be the number of occurrences of commutative operators in term t. First order the terms such that *C-Oprs*(t_1) $\geq$ *C-Oprs*(t_2). Then perform S-unification for each pair from $\{t_1\} \times$ *C-Permute*(t_2), with the slight modification that two terms be considered as identical whenever they are C-equal. These modifications greatly improve performance over the obvious solution, while maintaining finiteness and completeness. The algorithm is still not minimal, however. The complete C-unification algorithm is given in Figure 3.

```
Procedure C-Unify (t1, t2, Σ)
    If C-Oprs(t2) > C-Oprs(t1)
    Then  t1, t := Swap(t1, t2)
    Return ∪_{s∈ C-Permute(t)} S-Unify(t1, s, Σ)
End Procedure C-Unify
```

Figure 3. Commutative Unification Algorithm

The following example illustrates the use of C-Unify:

Example 3: Suppose we want to unify $f(a, x)$ and $f(y, f(z, a))$, modulo commutativity. We begin with the call *C-Unify*($f(a, x)$, $f(y, f(z, a))$, $\{\phi\}$). The terms are already properly ordered, thus permutations are found by

$$C\text{-}Permute(f(y, f(z, a))) = \{f(y, f(z, a)), f(f(z, a), y), f(y, f(a, z)), f(f(a, z), y)\}.$$

This results in the following calls to *S-Unify*:

(1) *S-Unify*($f(a, x)$, $f(y, f(z, a))$, $\{\phi\}$) returns $\{\{y \leftarrow a,\ x \leftarrow f(z, a)\}\}$

(2) *S-Unify*($f(a, x), f(f(z, a), y), \{\phi\}$) fails and returns ϕ

(3) *S-Unify*($f(a, x), f(y, f(a, z)), \{\phi\}$) returns $\{\{y \leftarrow a, \ x \leftarrow f(a, z)\}\}$

(4) *S-Unify*($f(a, x), f(f(a, z), y), \{\phi\}$) fails and returns ϕ

Thus *C-Unify* returns $\{ \{y \leftarrow a, \ x \leftarrow f(z, a)\}, \{y \leftarrow a, \ x \leftarrow f(a, z)\} \}$.

3. Associative/Commutative Unification.

In this section we will jointly address both AC-unification and ACI-unification. We first examine the problem of ACI-unification for two terms which have the same root ACI operator. We then consider the problem of ACI-unification of two terms which do not have the same root ACI operator. Finally, we show how AC-unifiers can be generated from ACI-unifiers.

a. ACI-Unification: Same Root Operator. The algorithm presented here is due to Stickel [*St*81]. Before beginning the ACI-unification process, terms are flattened according to the method described earlier in this chapter. Those terms which contain operands which are not variables are then converted into terms with only variables as operands, introducing new variables where necessary and recording the substitution necessary to undo this change at a point later in the process. This process, called *variable abstraction*, is illustrated in the following example: Suppose we want to ACI-unify the terms $f(x_1, a)$ and $f(y_1, y_1)$ where f is an ACI operator with identity e, x_1 and y_1 are variables, and a is a constant. Performing variable abstraction we generate the new variable x_2, and the new *variable only* terms, $f(x_1, x_2)$ and $f(y_1, y_1)$. We record the substitution $\sigma_1 = \{x_2 \leftarrow a\}$ for later use.

We now address the case of ACI-unification for two terms which begin with the same ACI operator and contain only variables in the scope of that operator. Stickel's algorithm is as follows:

(1) Eliminate common operands.

(2) Form an equation from the two terms where the coefficient of each variable in the equation is equal to the multiplicity of the corresponding variable in the term.

(3) Generate all non-negative integral solutions to the equation, eliminating all those solutions composable from other solutions.

(4) Associate a new variable with each solution. These will be called the *introduced variables.*

(5) Assemble a single unifier composed of assignments to the original variables with as many of each new variable as specified by the solution element in the sum associated with the new variable and the original variable. Zero components in the solution represent an assignment to the identity.

Resuming our example, we now apply the above algorithm to our variable only terms $f(x_1, x_2)$ and $f(y_1, y_1)$:

(1) There are no common operands to remove for this example.

(2) The equation to be solved is $x_1 + x_2 = 2y_1$. The class of equations which arise at this step is called the class of *homogeneous linear diophantine equations.* An algorithm for finding the basis of solutions for such equations is given by Huet [*Hu*78].

(3) The basis of solutions is:

Solution	x_1	x_2	y_1	$x_1 + x_2$	$2y_1$	New Variable
1	0	0	0	0	0	z_1
2	0	2	1	2	2	z_2
3	1	1	1	2	2	z_3
4	2	0	1	2	2	z_4

(4) The introduced variables, z_i, are shown in the table above.

(5) The single ACI-unifier for the variable only terms may be read from the columns of the above table and is given by

$$\sigma_2 = \{x_1 \leftarrow f(z_3, z_4, z_4), x_2 \leftarrow f(z_2, z_2, z_3), y_1 \leftarrow f(z_2, z_3, z_4)\}.$$

For terms which are not variable only terms, we must now reconcile the unifier from the variable only terms with the substitution recorded during the variable abstraction step. All possible reconciliations must be considered. It is at this point that our single unifier from the variable only case may give us a set of unifiers. For our example we must reconcile σ_1 and σ_2. Combining $x_2 \leftarrow a$ and $x_2 \leftarrow f(z_2, z_2, z_3)$ yields $z_2 \leftarrow e$ and $z_3 \leftarrow a$. Since there are no other variables replaced in σ_1 we may quit and

apply the result of the reconciliation to σ_2, giving us the single unifier $\{x_1 \leftarrow f(a, z_4, z_4),\ x_2 \leftarrow a, y_1 \leftarrow f(a,z_4)\}$. As the variable x_2 does not appear in either of the original terms to be unified, it may be dropped, giving us a final unifier of $\{x_1 \leftarrow f(a, z_4, z_4),\ y_1 \leftarrow f(a,z_4)\}$.

It is important to note that if there had been other substitution pairs in σ_1 in the previous example, we would have needed to continue reconciling the reconciliations. For example, ACI-unifying $f(x,a)$ and $f(b,b)$ yields exactly the same problem as the previous example after variable abstraction, except that $\sigma_1 = \{x_2 \leftarrow a, y_1 \leftarrow b\}$. In this case the reconciliation of σ_1 and σ_2 proceeds as follows:

(1) Combining $x_2 \leftarrow a$ and $x_2 \leftarrow f(z_2, z_2, z_3)$ yields $z_2 \leftarrow e$ and $z_3 \leftarrow a$.

(2) Combining $y_1 \leftarrow b$ and $y_1 \leftarrow f(z_2, z_3, z_4)$ yields three possibilities:
(a) $z_2 \leftarrow b,\ z_3 \leftarrow e,\ z_4 \leftarrow e$
(b) $z_2 \leftarrow e,\ z_3 \leftarrow b,\ z_4 \leftarrow e$
(c) $z_2 \leftarrow e,\ z_3 \leftarrow e,\ z_4 \leftarrow b$

(3) Reconciling (2a) with (1) fails because $z_2 \leftarrow e$ and $z_2 \leftarrow b$ conflict.
Reconciling (2b) with (1) fails because $z_3 \leftarrow a$ and $z_3 \leftarrow b$ conflict.
Reconciling (2c) with (1) fails because $z_3 \leftarrow a$ and $z_3 \leftarrow e$ conflict.

(4) As no possibility from (2) will reconcile with (1), no ACI-unifier is possible.

Figure 4 summarizes Stickel's ACI-unification algorithm for terms with the same root operator. This algorithm is shown to be finite, complete, and minimal whenever there are no other operators imbedded in the terms [*St*81,*Fa*84].

b. ACI-Unification: Different Root Operators..

Because of the identity equation, ACI-unification is possible between two terms which have different root operators. This can only happen when at least one of the root operators is an ACI operator which appears in a very special context. Suppose we have the terms $t_1 = x + (a*b)$ and $t_2 = (u*v)$, where + and * are ACI operators with identities 0 and 1, respectively. It is possible to *collapse* the outer level of t_1 by applying the substitution $\{x \leftarrow 0\}$ and then moving to another member of the

Procedure **ACI-Unify-Same** (t_1, t_2, Σ)

$t_1, t_2 := Eliminate\text{-}Common(t_1, t_2)$

For each $(s_1, s_2) \in Make\text{-}ACI\text{-}Pairs(t_1, t_2)$ do

$\Sigma := \cup_{\sigma \in \Sigma} S\text{-}Unify(s_1\sigma, s_2\sigma, \{\sigma\})$

End For

Return Σ

End Procedure ACI-Unify-Same

Procedure **Make-ACI-Pairs** (t_1, t_2)

$s_1, m_1 := Multiplicities(t_1)$

$s_2, m_2 := Multiplicities(t_2)$

$b_1, b_2 := Basis(m_1, m_2, t1.root)$

Return $\{(s_i\ b_i) \mid s_i \in s_1 \text{ and } b_i \in b_1\} \cup \{(s_i\ b_i) \mid s_i \in s_2 \text{ and } b_i \in b_2\}$

End Procedure Make-ACI-Pairs

Notes:

$Eliminate\text{-}Common(x + y + a,\ a + b)$ returns $t_1 = x + y$ and $t_2 = b$

$Multiplicities(x + y + a + y + a)$ returns $s = (x,\ y,\ a)$ and $m = (1,\ 2,\ 2)$

$Basis((1,\ 1),\ (2),\ +)$ returns $(z_3 + z_4 + z_4,\ z_2 + z_2 + z_3),\ (z_2 + z_3 + z_4)$ (see example)

Figure 4. ACI-Unification: Same Root Operator

resulting ACI-congruence class, namely $(a*b)$. At this point we have effectively removed + as the root operator of the term and replaced it with $*$, which can now match the root operator of t_2. It is easy to see that this can only happen when all or

all but one of the first level operands of an ACI operator are variables. If all of the first level operands are variables and there are n first level operands, then the term may be collapsed n different ways, with each variable unifying with t_2. When all but one of the first level operands are variables, the only possible unification is to set all the variables to the identity and then ACI-unify the remaining operand with t_2. When the two terms are rooted with different operators, both of which are ACI, the collapsing process must be attempted in both directions. For our example, collapsing t_2 allows both u and v to unify with all of t_1. Mayfield's algorithm for ACI-unification which handles two terms with differing root operators is given in Figure 5. The finiteness and completeness of this algorithm are addressed in [*Ma88*].

Procedure **ACI-Unify** (t_1, t_2, Σ)

{Assumes that one or both terms have an ACI operator at the root}

Case

1: $t_1 \stackrel{E}{=} t_2 \Rightarrow$ Return Σ

2: *Is-Var*$(t_1) \Rightarrow$ If t_1 occurs in t_2
Then Return ϕ
Else Return $\cup_{\sigma \in \Sigma} \sigma \circ \{t_1 \leftarrow t_2\}$

3: *Is-Var*$(t_2) \Rightarrow$ Return *ACI-Unify*(t_2, t_1, Σ)

4: $\neg$ *Is-ACI*$(t_1.root) \Rightarrow$ Return *ACI-Unify-Diff*(t_1, t_2, Σ)

5: $\neg$ *Is-ACI*$(t_2.root) \Rightarrow$ Return *ACI-Unify-Diff*(t_2, t_1, Σ)

6: $t_1.root \neq t_2.root \Rightarrow$ Return *ACI-Unify-Diff*$(t_1, t_2, \Sigma) \cup$
ACI-Unify-Diff(t_2, t_1, Σ)

7: Otherwise $\Rightarrow$ Return *ACI-Unify-Same*(t_1, t_2, Σ)

End Procedure ACI-Unify

Figure 5A. ACI-Unification Algorithm

Procedure **ACI-Unify-Diff** (t_1, t_2, Σ)

{Assumes t_2.*root* is an ACI operator, $+$, with identity i}

{Assumes t_1 is either simple or has a different operator}

$n := \mid Fdom(t_2) \mid$

$I := i + i + \ldots + i$ {n identities}

$\Sigma' := \phi$

For $j := 1$ to n do

$t'_1 := I[j \leftarrow t_1]$

$\Sigma' := \Sigma' \cup S\text{-}Unify(t'_1, t_2, \Sigma)$

End For

Return Σ'

End Procedure ACI-Unify-Diff

Figure 5B. ACI-Unification: Different Root Operators

c. AC-Unification via ACI-Unification.

Stickel [*St*81] also suggests a method for generating AC-unifiers from ACI-unifiers. To do this we begin by treating the AC operators as if they were ACI operators and generating a complete set of ACI-unifiers. We then substitute the identity for introduced variables in all possible combinations in the right hand side of substitution pairs for each ACI-unifier. This is subject to the restriction that no unifier is retained which assigns one of the original variables to the identity. Suppose the operator f had been an AC operator in the previous example where we ACI-unified $f(x_1, a)$ and $f(y_1, y_1)$. The ACI-unifier for these terms was

$\{x_1 \leftarrow f(a,z_4,z_4),\ y_1 \leftarrow f(a,z_4)\}$. The only remaining introduced variable is z_4. We can now either substitute the identity for z_4 or leave z_4 alone, giving

$$\{\ \{x_1 \leftarrow a,\ y_1 \leftarrow a\},\ \{x_1 \leftarrow f(a,z_4,z_4),\ y_1 \leftarrow f(a,z_4)\}\ \}$$

as the complete set of AC-unifiers for our terms.

The AC-unification algorithm described is given in Figure 6. This algorithm is shown to be finite and complete in [*Wi87*].

```
Procedure AC-Unify (t1, t2, Σ)
    Σ := ACI-Unify(t1, t2, Σ)
    Σ' := φ
    For each σ ∈ Σ do
        Z := Vars(σ) − Vars(t1) − Vars(t2)
        {Let i be a temporary identity for t1.root}
        α := {{z ← i} | z ∈ Z}
        For each γ ∈ 2^α do
            If (v ← t) ∈ (σγ)↓' ⇒ t ≠ i
            Then Σ' := Σ' ∪ (σγ)↓'
        End For
    End For
End Procedure AC-Unify
```

Figure 6. AC-Unification Algorithm

4. Combining E-Unification Algorithms.

Because of the nature of the algebraic systems over which we wish to find complete sets of reductions, we must deal with E-unification for two terms which contain various operators, each associated with its own equational theory. For example, we might want to E-unify the term $x + ((-y)*z)$ with $b*(-f(u,v))$, where $+$ is an ACI operator, $*$ is an AC operator, f is a C operator, and $-$ is an empty E operator.

Yellick [*Ye*85] has developed a framework for combining unification algorithms for equational theories which are both confined and regular. All of our candidate equational theories meet her definitions of these criteria except those which are ACI. These fail because the identity law is not confined. Mayfield [*Ma*88] has developed an interleaving of empty E, C, AC, and ACI unification which is a variation on the Yellick model. The basic approach of this method is that a top level E-unification procedure classifies the type of unification problem based on the type of operators at the roots of the two terms to be E-unified. After classification a call is made to the appropriate specialized E-unification routines which then make recursive calls back to the top level for the E-unification of any lower level operands. Figure 7 gives Mayfield's interleaved E-unification algorithm. This method is believed to be finite and complete for the interleaving of these four equational theories.

Procedure **E-Unify** (t_1, t_2, Σ)

{This handles ϕ, C, AC, and ACI equational theories}

Case

1: *Simple*(t_1) and *Simple*(t_2) ⇒ Return *S-Unify*(t_1,t_2,Σ)

2: *Simple*(t_1) ⇒ If *Is-ACI*(t_1.*root*)
Then Return *ACI-Unify*(t_1, t_2, Σ)
Else Return *S-Unify*(t_1, t_2, Σ)

3: *Simple*(t_2) ⇒ Return *E-Unify*(t_2, t_1, Σ)

4: *Is-C*(t_1.*root*) and *Is-C*(t_2.*root*) ⇒ Return *C-Unify*(t_1, t_2, Σ)

5: *Is-AC*(t_1.*root*) and *Is-AC*(t_2.*root*) ⇒ Return *AC-Unify*(t_1, t_2, Σ)

6: *Is-ACI*(t_1.*root*) and *Is-ACI*(t_2.*root*) ⇒ Return *ACI-Unify*(t_1, t_2, Σ)

7: Otherwise ⇒ Return *S-Unify*(t_1, t_2, Σ)

End Procedure E-Unify

Note: This requires that S-Unify and ACI-Unify-Same be modified to call back to E-Unify, rather than to S-Unify

Figure 7. Interleaved E-Unification Algorithm

III. COMPLETE SETS OF REDUCTIONS

Building on the definitions and concepts presented in the previous chapter, we now focus our attention more specifically on the theory of complete sets of reductions. We begin this chapter by giving formal definitions related to completeness. We then discuss the application and benefits of complete sets of reductions in the context of automated theorem proving. Finally, we review major contributions from the literature relative to the theory of generating complete sets of reductions.

A. DEFINITIONS

The *universal word problem* is the problem of deciding whether or not two arbitrary algebraic terms are equal with respect to a given equational theory. This problem has been shown to be undecidable in the general case. We will address a class of instances where it is decidable.

A rewriting relation, $\xrightarrow{R}$, is said to be *noetherian* or *finitely terminating* if no infinite descending chain $t_1 \xrightarrow{R} t_2 \xrightarrow{R} t_3 \xrightarrow{R} \ldots$ exists. When a noetherian rewriting relation $\xrightarrow{R}$ is applied to a term t until it can be applied no more, the resulting irreducible term is called a *terminal form* or *normal form* of t, written as $t{\downarrow}^R$. Clearly there may be more than one terminal form for t, however, all such forms are computable whenever R is noetherian.

We say that a rewriting relation $\xrightarrow{R}$ on terms $T(F,V)$ has the *Church-Rosser property* whenever for all terms t_1, $t_2 \in T(F,V)$, $t_1 \overset{R}{=} t_2$ implies that there exists another term $t_3 \in T(F,V)$ such that $t_1 \xrightarrow{R}{}^{*} t_3$ and $t_2 \xrightarrow{R}{}^{*} t_3$. An alternate way of stating this concept is to say that equivalent terms have a common rewriting under $\xrightarrow{R}$. Whenever $\xrightarrow{R}$ is a noetherian rewriting relation which satisfies the Church-Rosser property, the set R is

called a *complete set of reductions.* It is easy to see that whenever R is a complete set of reductions, each term has a unique terminal form. Suppose t_1 and t_2 are two terms which are equivalent under the E-equality relation generated by R. By the finite termination property of R we can find terminal forms $t_1\downarrow^R$ and $t_2\downarrow^R$. Clearly these terminal forms are equivalent under R, and thus, by the Church-Rosser property they must have a common rewriting. But since they are terminal forms, they cannot be rewritten, thus they must already be identical. We call these unique terminal forms *canonical forms.* An equivalent alternate definition of completeness is that R is complete whenever $t_1 \overset{R}{=} t_2 \Rightarrow t_1\downarrow^R = t_2\downarrow^R$.

B. USING COMPLETE SETS OF REDUCTIONS

Before examining the manner in which complete sets of reductions are generated, let us examine our motivations for generating them in the first place. We first address how they may be used and then the efficiency benefits which they bring.

1. Applications.

We now point out that the existence of a complete set of reductions implies a solution to the universal word problem for the relevant domain. In order to determine whether or not two terms are equivalent with respect to a given equational theory for which a complete set of reductions exists, we simply find the canonical forms for each term and compare them. If the canonical forms are identical, then the terms are equivalent, otherwise they are not. This constitutes a decision procedure for the word problem. Because of the strength of this property, complete sets of reductions may be used as (1) canonical simplifiers, (2) the basis for proving theorems, and (3) an augmentation for other proof techniques.

When a theorem to be proved is of the form $t_1 \stackrel{A}{=} t_2$ where a complete set of reductions exists for the algebraic system, A, then we can prove or disprove the theorem by deciding the word problem as described above. Even when a theorem to be proved is not of the form $t_1 \stackrel{A}{=} t_2$, and another inference technique must be used to attempt a proof, a complete set of reductions may be very useful. For example, suppose we are using the resolution rule of inference [*Ro*65] to attempt a proof of the theorem P in a system where we have among the axioms a rule such as *IF* $Q(t_1)$ *THEN* P. Suppose further that the resolution mechanism has just generated the clause $Q(t_2)$ where t_1 and t_2 are not unifiable. Now if a complete set of reductions can be used to reduce t_1 to t'_1 and t_2 to t'_2 where t'_1 and t'_2 will unify, then the resolution mechanism can conclude P. Thus a complete set of reductions can be used to augment resolution and similarly any other inference technique.

Hullot [*Hu*80] has catalogued several problem domains for which complete sets of reductions have been found, thus making them candidates for these applications.

2. Efficiency Benefit.

The primary advantage of using a complete set of reductions in automated theorem proving is one of efficiency. It is theoretically possible to prove with resolution anything which can be proved by applying a complete set of reductions. It is also often possible to prove these same results using incomplete sets of rewrite rules and/or reductions. Yet none of these other techniques can compete with complete sets of reductions in terms of efficiency.

One of the greatest gains in efficiency over resolution comes by virtue of the fact that complete sets of reductions are rewrite rules and thus are more suited for dealing with the equality relation. Soon after Robinson introduced the concept of resolution, it was realized that resolution is very inefficient when it comes to dealing with the

equality relation. Siebert [*Si*68] pointed out early on that proof procedures for logic systems with equality suffer greatly when they must treat equality as any other binary relation, with axioms added to give it desired properties such as reflexivity, symmetry, transitivity, and equality under argument substitution. Similarly, Robinson and Wos [*WR*69] stated that the intermediate debris generated in applying equality axioms with resolution generate increasingly larger generations of useless offspring, polluting the search space badly. They conclude that a substitution rule of inference, or rewrite rule, tends to be more convergent. We might say that rewrite rules, which in essence have the equality axioms built in to the inference mechanism, seem to capture the semantics of the equality relation whereas resolution has only a syntactic level encoding of the equality relation.

The application of a complete set of reductions provides the greatest gains in efficiency over resolution and the application of incomplete reduction sets in the area of the search mechanism. These gains in efficiency come from the completeness property. Whereas resolution, rewrite, and reduction inference mechanisms must expand their search trees in some breadth first fashion in order to be able to guarantee that they will not miss a proof, applying a complete set of reductions completely eliminates the need for a search tree. Since every path through the tree leads to the exact same canonical form, the search tree collapses to a linear path, with every step producing a result that is guaranteed to be closer to the final solution. Thus the reductions may be applied in any order. This elimination of a complicated search results in a greatly simplified algorithm for applying a complete set of reductions, while at the same time significantly reducing both the time and space needed for execution.

A simple example will serve to illustrate the efficiency gains which we have discussed. For this example we use the complete set of ten reductions generated by Knuth and Bendix [*KB*70] for group theory to prove a simple identity. The

reductions are used first as a reduction set and then together with equality axioms in a resolution system. The Interactive Theorem Prover, ITP, from Argonne Labs [*OL*84] was used for both runs. The problem was to prove the identity $(e*a)^{-1}*(a*e) = (e^{-1})^{-1}$.

When ITP was given this problem using only demodulation (application of reductions), it was able to find the solution by way of applying five reductions and making 64 attempted term matches. When ITP was given this same problem plus the necessary equality axioms, using full binary resolution, weighting strategies, subsumption, and the set of support strategy, it generated 123 clauses and attempted 17,585 unifications before finding an eight step proof of the identity. Here the combined effects of dealing poorly with the equality relation and having to expand the search tree can clearly be seen.

Note that these gains are multiplied every time we use a complete set of reductions, whereas the cost of generating a complete set of reductions is a one time charge. Once we have a complete set of reductions for a given algebraic system we need never generate it again, yet we can use it over and over as we attempt to prove theorems relative to the domain of the given algebraic system.

C. GENERATING COMPLETE SETS OF REDUCTIONS

We now address the process whereby a complete set of reductions may be generated. We refer to such procedures as *completion procedures.* We will see that each of the completion procedures presented is actually a very slight generalization of another type of procedure, a procedure which tests a given set of reductions for completeness. Because of this researchers have generally approached the problem of developing a completion procedure by first developing testable conditions which imply completeness. Keeping this in mind we now review what we believe to be the

more important contributions from the literature relative to the development and generalization of completion procedures. The research which we will present in subsequent chapters will build on concepts developed in each of these earlier works.

1. Knuth-Bendix Completion Procedure.

In 1967 Knuth and Bendix [*KB70*] presented an algorithm for determining whether or not a given set of reductions is complete and a procedure which may be able to complete an incomplete set of reductions. Their theory centers around the development of testable properties which imply the finite termination and Church-Rosser properties, thereby implying completeness.

a. Testing the Finite Termination Property.

The method used by Knuth and Bendix to establish the finite termination property is to establish a *weighting function* and thereby an *ordering relation* on the set of all terms to be considered. Suppose we can define a weighting function which assigns a positive integer weight to any term which we wish to consider and that we can show that the weight of a term strictly decreases each time a reduction is applied. Since the weight of a term can never be at or below zero, there can only be a finite number of applications of the rewrite rules before we reach a terminal form. Let $W(t)$ be a function which gives the weight of term t. The key here is defining the weighting function in such a way that the weight of a term *strictly decreases* each time a reduction is applied. This requirement can then be shown whenever $W(\lambda) > W(\rho)$ for all reductions in the reduction set and the following two properties are provided by the ordering relation:

(1) $W(t_1) > W(t_2) \Rightarrow W(t_1\sigma) > W(t_2\sigma)$ for all substitutions σ. (Ordering is preserved under the application of substitutions.)

(2) $W(t_1) > W(t_2) \Rightarrow W(f(....t_1...)) > W(f(...t_2...))$. (Terms differing only by a subterm have their ordering determined by the ordering of their differing subterms.) We say that the weighting function is *compatible* with the term structure.

Recalling that a reduction is applied to a term $t \equiv t[m \leftarrow \lambda\sigma]$ producing a new term $t' \equiv t[m \leftarrow \rho\sigma]$, we can see that, given $W(\lambda) > W(\rho)$, then $W(\lambda\sigma) > W(\rho\sigma)$ by (1) and $W(t) > W(t')$ by (2). Thus applying the reduction will always decrease the weight. Knuth and Bendix go into great detail in their paper to define one such ordering relation and prove that it has the stated properties for terms of a certain structure. It is important to note, however, that any ordering relation with the required properties will suffice.

The ordering relation provides us with a way of testing for the termination property. The test for termination is really quite simple. Once we have a weighting function which meets the required properties, all we need to do is verify that $W(\lambda) > W(\rho)$ for every $\lambda \rightarrow \rho \in R$. If every reduction passes this test, then the finite termination of $\overset{R}{\rightarrow}$ is assured.

b. Testing the Church-Rosser Property.

In order to explain the test for the Church-Rosser property we will first present a series of new properties which imply this property. The following discussion does not follow precisely the same path as the original Knuth-Bendix paper, however, we believe it makes more clear the important steps leading up to the same point.

A rewriting relation, $\overset{R}{\rightarrow}$, is said to be *confluent* if for all terms t, t_1, and $t_2 \in T(F,V)$ $t \overset{R}{\underset{\bullet}{\rightarrow}} t_1$ and $t \overset{R}{\underset{\bullet}{\rightarrow}} t_2 \Rightarrow$ there exists a term $t_3 \in T(F,V)$ such that $t_1 \overset{R}{\underset{\bullet}{\rightarrow}} t_3$ and $t_2 \overset{R}{\underset{\bullet}{\rightarrow}} t_3$. This definition is illustrated in Figure 8.

It is easy to show that a noetherian rewriting relation $\overset{R}{\rightarrow}$ has the Church-Rosser property iff $\overset{R}{\rightarrow}$ has the confluence property. See Bundy [*Bu*83] for a simple proof of

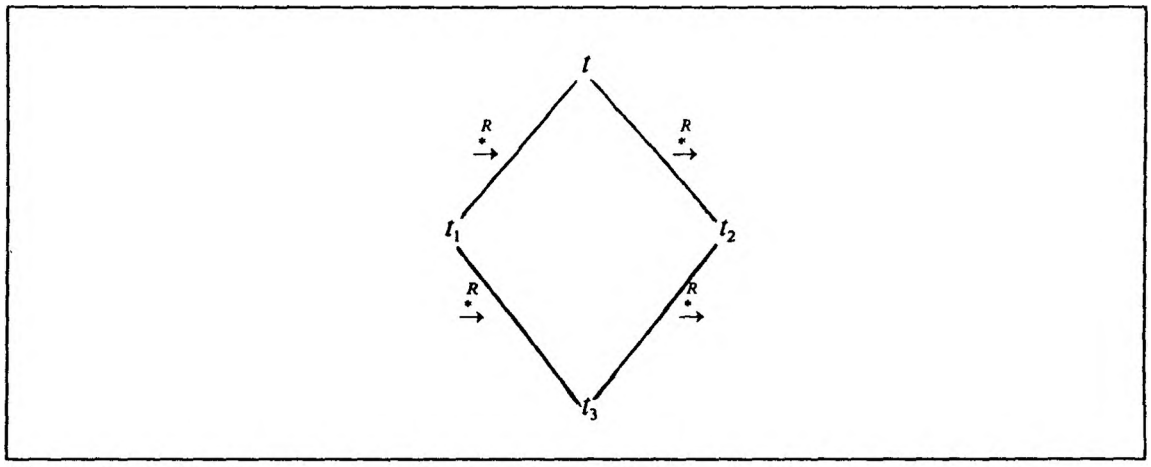

Figure 8. Confluence

this. Terms t_1 and t_2 are said to *conflate* whenever they have a common rewriting as in Figure 8.

We say that a rewriting relation, $\xrightarrow{R}$, is *locally confluent* if for all terms t, t_1, and $t_2 \in T(F,V)$ $t \xrightarrow{R} t_1$ and $t \xrightarrow{R} t_2 \Rightarrow$ there exists a term $t_3 \in T(F,V)$ such that $t_1 \xrightarrow{R}_{*} t_3$ and $t_2 \xrightarrow{R}_{*} t_3$. Note that this definition differs from the definition of confluence only in that t_1 and t_2 are each derived from t via a single application of $\xrightarrow{R}$. This definition is illustrated in Figure 9.

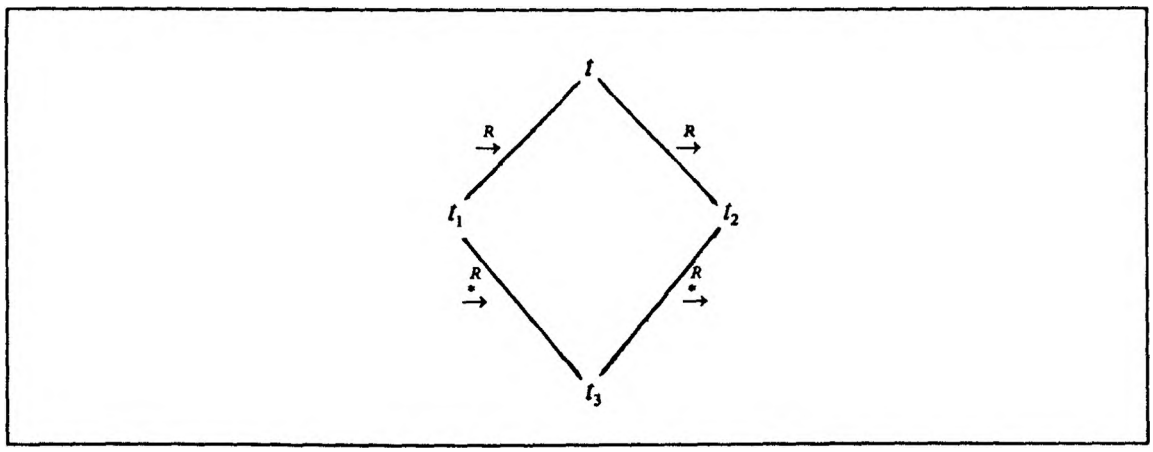

Figure 9. Local Confluence

It can be shown that finite termination and confluence imply local confluence. This was proposed by Newman in 1942 and proved in full generality by Huet in 1977 using a technique called noetherian induction [*Hu*81]. Bundy [*Bu*83] also has a nice summary of this proof.

Tracing back the chain of properties, we now see that the properties of finite termination and local confluence will assure us that we have a complete set of reductions. While the terms Church-Rosser, confluence, and local confluence do not appear in the original Knuth-Bendix paper, the result that termination and local confluence imply completeness is precisely the same concept which is expressed as the "lattice condition" in Theorem 4 of that paper. It is the property of local confluence which allowed Knuth and Bendix to design an algorithm to test a set of reductions for completeness.

The problem that remains is in showing that a set of reductions is locally confluent for a possibly infinite set of terms. It is in the design of a test for local confluence that Knuth and Bendix brought real insight to the problem. Rather than attempt to develop a test which operates on a possibly infinite set of terms, they developed a test which operates on the finite set of reductions which can be applied to those terms. This is very similar to the manner in which Robinson moved from examining interactions among an infinite number of instantiations of a set of rules to examining the interactions among a finite number of rules themselves when he developed the concepts of resolution and unification [*Ro*65].

Knuth and Bendix observed that local confluence can only be an issue when the reductions allow a term to be rewritten in more than one way. Suppose we have a term t which can be rewritten into t_1 by r_1 and into t_2 by r_2. They noted that either (1) r_1 and r_2 apply to totally different subterms of t, or (2) one rewritten subterm is entirely enclosed in the other rewritten subterm. (The fact that a term can be

represented as a tree prevents two subterms of the same term from overlapping each other only partially.)

In the first case r_1 may still be applied to t_2 to get t_3 and r_2 may still be applied to t_1 to get t_3, thus local confluence is preserved. In the second case the analysis is more involved: application of one rule may prevent the application of the other. Knuth and Bendix observed, however, that in this case there must be some subterm $t/m = \lambda_1\sigma_1$ and some subterm $\lambda_1\sigma_1/n = \lambda_2\sigma_2$. When r_1 and r_2 (and thus σ_1 and σ_2) are variable disjoint it can be shown that there exists a position $n' \in Dom(\lambda_1)$ such that $\lambda_1\sigma_1/n = (\lambda_1/n')\sigma_1$, giving $(\lambda_1/n')\sigma_1\sigma_2 = \lambda_2\sigma_1\sigma_2$ and $\sigma_1\sigma_2$ is a unifier for λ_1/n' and λ_2. This means that it is possible to find a most general unifier, θ, for λ_1/n' and λ_2.

Furthermore, we can determine what the resulting terms t_1 and t_2 will look like after r_1 and r_2 have been applied. By the definition of $\xrightarrow{R}$ we know that $t_1 = t[m \leftarrow \rho_1\sigma_1]$ and $t_2 = t[m \leftarrow \lambda_1\sigma_1[n \leftarrow \rho_2\sigma_2]]$. Clearly t_1 and t_2 are conflatable when their subterms $t_1/m = \rho_1\sigma_1$ and $t_2/m = \lambda_1\sigma_1[n \leftarrow \rho_2\sigma_2]$ are conflatable. Noting that $t_1/m = \rho_1\sigma_1\sigma_2$ and $t_2/m = (\lambda_1[n' \leftarrow \rho_2])\sigma_1\sigma_2$ when r_1 and r_2 are variable disjoint, it is clear that $t_1/m = \rho_1\sigma_1\sigma_2$ is an instance of $c_1 = \rho_1\theta$ and $t_2/m = (\lambda_1[n' \leftarrow \rho_2])\sigma_1\sigma_2$ is an instance of $c_2 = (\lambda_1[n' \leftarrow \rho_2])\theta$ where θ is the most general unifier of λ_1/n' and λ_2, as described above. It follows that t_1/m and t_2/m are conflatable whenever c_1 and c_2 are conflatable. Although the term does not appear in the original Knuth-Bendix paper, the pair $< c_1, c_2 >$ has generally been called the *critical pair* of the Knuth-Bendix algorithm.

We can now detect all situations of this type by attempting to unify all of the subwords of all λs with all other λs in the set of reductions. (This is called the "superposition" process in the Knuth-Bendix paper.) When the unification of λ_1/n with λ_2 succeeds yielding a unifier, θ, we can then form the critical pair $< \rho_1\theta,\ \lambda_1[n \leftarrow \rho_2]\theta >$. For example, when $\lambda_1 \rightarrow \rho_1$ is $(x*y)*z \rightarrow x*(y*z)$ and $\lambda_2 \rightarrow \rho_2$ is

$u^{-1}*u \rightarrow e$, the subterm $(x*y)$ of λ_1 unifies with λ_2 yielding $\theta = \{x \leftarrow u^{-1},\ y \leftarrow u\}$. This gives the critical pair $< (x*(y*z))\theta,\ (e*z)\theta >$, or, after applying the substitution, $< (u^{-1}*(u*z)),\ e*z >$.

This now provides a method for testing a terminating set of reductions for local confluence. For every pair $(r_1,\ r_2) \in R \times R$ the critical pairs are found as described above. Then for each critical pair $< c_1,\ c_2 >$ we compute the terminal forms $c_1{\downarrow}^R$ and $c_2{\downarrow}^R$. If the terminal forms are identical, then the pair is conflatable, otherwise it is not. If all critical pairs conflate, then local confluence is assured and the reductions form a complete set.

c. A Completion Procedure. The algorithm described for testing a set of reductions for completeness suggests a procedure for possibly extending an incomplete set of reductions to make it complete. Suppose we are testing a critical pair $< c_1,\ c_2 >$, formed from the reduction set R, for conflatability by comparing terminal forms $c_1{\downarrow}^R$ and $c_2{\downarrow}^R$, and we find that these terms are not identical. By the very nature of the critical pair process we know that $c_1{\downarrow}^R$ and $c_2{\downarrow}^R$ are clearly equivalent with respect to the equational theory represented by R. If these non-conflating terms can be ordered properly so as to preserve the finite termination property, then we can form a new reduction, either $c_1{\downarrow}^R \rightarrow c_2{\downarrow}^R$ or $c_2{\downarrow}^R \rightarrow c_1{\downarrow}^R$, depending on which terminal form has greater weight according to the term weighting function. Adding this new reduction "forces" the confluence of the troublesome critical pair. If, however, neither term has more weight than the other according to our ordering relation, we cannot make the pair conflatable and we must terminate with failure. Of course, every time we add a new reduction to R we introduce the possibility of new critical pairs and thus the process must be repeated until on a single pass all critical pairs are found to conflate without the addition of any new reductions. The critical pair step is often called the *inference step* of the completion

process because it is out of that step that new reductions are generated which bring the set of reductions closer to completeness.

With the addition of new reductions there is also the possibility that some of the old reductions are no longer needed to have a complete set. After a new reductions is added to R, for all old reductions $\lambda \rightarrow \rho$, λ and ρ can be reduced to their terminal forms by the set of reductions $R - \{\lambda \rightarrow \rho\}$. If $\lambda\downarrow^{R-\{\lambda\rightarrow\rho\}}$ and $\rho\downarrow^{R-\{\lambda\rightarrow\rho\}}$ are identical then $\lambda \rightarrow \rho$ is not needed to retain completeness and may be deleted from R. This simplification step is not necessary, but may be used to generate a *minimal* complete set of reductions.

The Knuth-Bendix completion procedure given in Figure 10 is an adaptation of one given by Musser and Kapur [*MK*82].

```
Procedure Completion (R)
   Repeat
      R' := R
      For each < c1, c2 > ∈ U(p,q)∈R×R Critical-Pairs(p, q) do
         If c1↓R ≠ c2↓R
         Then
            Case
               W(c1↓R) > W(c2↓R):  R := R ∪ {c1↓R → c2↓R}
               W(c2↓R) > W(c1↓R):  R := R ∪ {c2↓R → c1↓R}
               Otherwise Halt with Failure
         End If
         {R may be simplified here, if desired}
      End For
   Until R = R'
End Procedure Completion

Procedure Critical-Pairs (λ1 → ρ1, λ2 → ρ2)
   Return
   { < ρ1σ, ρ2σ >  |  σ = S-Unify(λ1, λ2)} ∪
   { < ρ1σ, (λ1[m ← ρ2])σ >  |  m ∈ Sdom(λ1), m ≠ ε,  and σ = S-Unify(λ1/m, λ2)} ∪
   { < (λ2[m ← ρ1])σ, ρ2σ >  |  m ∈ Sdom(λ2), m ≠ ε,  and σ = S-Unify(λ1, λ2/m)}
End Procedure Critical-Pairs
```

Figure 10. A Knuth-Bendix Completion Procedure

There are three possible outcomes from this completion procedure: (1) it may halt after finding a complete set of reductions, (2) it may halt with failure because a non-conflatable pair cannot be ordered to form a new reduction, or (3) it may find critical pairs and add new reductions on every iteration, thus never halting. When the procedure fails because of an ordering problem it may be possible to try again with another term weighting function. Because any ordering relation which meets the properties stated earlier will suffice, it is often possible to find a different ordering relation which properly orients the offending pair. Then we may start over at the beginning of the procedure.

Knuth and Bendix were able to generate complete sets of reductions for some small algebraic systems using this procedure. The most notable of these is the system for a free group where they were able to start with a set of three equations and generate a complete set of ten reductions. The complete proof of correctness for the Knuth-Bendix completion procedure was given by Huet in 1977 [*Hu*81].

It is interesting to note that the Knuth-Bendix completion procedure can be thought of as one instance in the general class of of *critical pair completion procedures* which includes many other well known procedures, such as Euclid's algorithm. See Buchberger [*Bu*85] for an interesting comparison of the Knuth-Bendix procedure to other procedures in this class.

2. Peterson-Stickel E-Completion Procedure.

As we pointed out in the introductory chapter, the Knuth-Bendix completion procedure is able to generate complete sets of reductions for a limited number of algebraic systems. Completion procedures based on the original Knuth-Bendix theory

are not able to handle any algebraic system whose definition includes a commutativity axiom because these axioms cannot be oriented to form a reduction. Thus the finite termination property cannot be maintained. Peterson and Stickel [*PS*81] were able to overcome this limitation of completion procedures by splitting the equational axioms of an algebraic system into two sets: (1) equations which are incorporated in the matching and unification processes needed to apply reductions and compute critical pairs, and (2) equations which form the basis of the reduction set to be completed. This type of completion procedure has come to be called an *E-completion procedure* where E is the set of equations built into the matching and unification processes. We first review the features which set this theory apart from the Knuth-Bendix theory, and then present the E-completion procedure itself.

a. E-Unification and E-Matching Approach.

The most significant feature of the Peterson-Stickel E-completion procedure is the use of E-unification to compute critical pairs and E-matching to apply reductions. The combined effect of these two operations is that single inferences within the completion procedure are actually performed on entire E-congruence classes of terms, rather than just on single terms. Two E-equal terms are treated as if they are the same term and there is never any reason to rewrite a term into an E-equal term. Thus when we have an algebraic system with an unorientable axiom such as commutativity, building that axiom into the unification and matching processes prevents us from having to handle it as a reduction. This means that the Peterson-Stickel theory is applicable to entire classes of algebraic systems which cannot be addressed with the Knuth-Bendix theory.

Not only does this method of computing over E-congruence classes open up entire new problem domains for completion procedures, it provides a more general and more efficient manner for dealing with some equations which were handled as

reductions under the Knuth-Bendix theory. The associative law is an example of such an equation. Peterson and Stickel point out that, although it can be oriented as a reduction, some generality is lost in the process. Furthermore, when associativity and commutativity are built into E together, the E-congruence classes become even larger, allowing a single inference to cover more cases, resulting in greater efficiency.

The primary requirements for the Peterson-Stickel theory are (1) the existence of a finite and complete unification algorithm for the equational theory E, and (2) the finite termination property for the rewriting relation $\overset{R,E}{\rightarrow}$. Peterson and Stickel show that the existence of such an E-unification algorithm implies the existence of an E-matching algorithm and a decision procedure for E-equality, each of which are also required by the theory. An additional requirement is that the equations of E all contain exactly two occurrences of each variable, one on the left and one on the right.

b. E-Completeness and E-Compatibility.

In the Peterson-Stickel theory, an *E-complete set of reductions* is defined to be a set of reductions R such that for all pairs of terms t_1 and t_2, $t_1 \overset{R\cup E}{=} t_2 \Rightarrow t_1\downarrow^{R,E} \overset{E}{=} t_2\downarrow^{R,E}$. This generalization of the earlier definition of completeness says that R is E-complete when terms which are equivalent with respect to the entire algebraic system reduce to terminal forms in the same E-congruence class.

A set R of reductions is said to be *E-compatible* if whenever $t_1 \overset{R,E}{\rightarrow} t_2$, there exist a node $m \in Dom(t_1)$, terms t'_1, t'_2, a substitution σ, and a reduction $\lambda \rightarrow \rho \in R$ such that $t_1/m \overset{E}{=} \lambda\sigma$ and $t_2 \overset{E}{=} t'_2 \overset{R,E}{\underset{*}{\rightarrow}} t'_1 \overset{E}{=} t_1[m \leftarrow \rho\sigma]$.

If $r_1 \equiv \lambda_1 \rightarrow \rho_1$ and $r_2 \equiv \lambda_2 \rightarrow \rho_2$ are two reductions from R then *E-Critical-Pairs*(r_1, r_2) is defined to be the set of all pairs $< (\lambda_1[m \leftarrow \rho_2])\sigma,\ \rho_1\sigma >$ such that $m \in Sdom(\lambda_1)$, $\sigma \in$ *E-Unifiers*$(\lambda_1/m, \lambda_2)$. Note that E-critical pairs are computed

just like the critical pairs of the Knuth-Bendix theory, with the exception that E-unification is used in place of unification and multiple critical pairs may arise from one overlap because there is one pair per E-unifier.

E-Completeness Theorem: [*PS*81]

Let R be an E-compatible set of reductions and let $\xrightarrow{R}$ *be a rewriting relation satisfying the finite termination property. Then R is E-complete iff for every critical pair* $< c_1, c_2 > \in \cup_{(p,q)\in R\times R} E\text{-}Critical\text{-}Pairs(p, q)$ $c_1\downarrow^R \stackrel{E}{=} c_2\downarrow^R$.

In other words, E-completeness relies on the confluence of E-critical pairs modulo E-equality. The major problem with the E-completeness theorem is that R must be known to be E-compatible.

c. E-Compatibility and Extensions.

The following theorem presents sufficient conditions for E-compatibility.

E-Compatibility Theorem: [*PS*81]

Suppose E is an equational theory whose equations are linear and non-erasing and that R is a set of reductions. Suppose also that whenever $l = r \in E$ *(or* $r = l \in E$*),* $\lambda_1 \rightarrow \rho_1 \in R$, $m \in Sdom(l)$ *but* $m \neq \varepsilon$, *and* $\sigma \in E\text{-}Unify(l|m, \lambda_1)$, *it follows that there exist* $\lambda_2 \rightarrow \rho_2$ *and a substitution* γ *such that* $l[m \leftarrow \lambda_1] \stackrel{E}{=} \lambda_2\gamma$ *and* $l[m \leftarrow \rho_1] \xrightarrow[*]{R,E} \rho_2\gamma$, *then R is E-compatible.*

At this point we focus on the class of problems where E is an AC equational theory. For equations such as the commutative law, E-compatibility is satisfied immediately since there is no $m \in Sdom(l)$ such that $m \neq \varepsilon$. For the associative law equations, Peterson and Stickel showed how to extend the set R to satisfy the E-compatibility requirements.

An *AC extension relative to E* of a reduction $\lambda \rightarrow \rho$, where + is an AC operator in E which is the root of λ, is a new reduction $v + \lambda \rightarrow v + \rho$ where v is a variable such that $v \notin Vars(\lambda \rightarrow \rho)$. Define R_E^{ac} to be the union of R and the set of all reductions which are AC extensions relative to E of reductions in R.

Extension Theorem: [*PS*81]

Let E be an AC equational theory and R be a set of reductions. Then R_E^{ac} *is E-compatible.*

This theorem states that we can maintain E-compatibility for AC theories by adding AC extensions for each reduction. Recall from the definition of AC extensions that they only exist when the root operator of the left side of a reduction is an AC operator. Putting this result together with the previous result that E-compatibility together with confluence modulo E-equality of E-critical pairs yields an E-complete reduction set leads to a modified version of the Knuth-Bendix completion procedure which is sufficient to perform AC-completion.

d. An AC-Completion Procedure.

Figure 11 presents an adapted version of the Peterson-Stickel AC-completion procedure. Using an implementation of this procedure, Peterson and Stickel were able to generate AC-complete reduction sets for a number of algebraic systems, among them commutative groups, commutative rings, and distributive lattices [*PS*82].

Procedure **AC-Completion** (CP)
  $RR := \phi$
  While $(RR \cup CP) \neq \phi$ do
    $R, RR :=$ *Try-To-Conflate-Pairs*(R,RR,CP)
    If $RR \neq \phi$
    Then $(r_1, r_2) :=$ "smallest" member of RR
      $RR := RR - \{(r_1, r_2)\}$
      $CP :=$ *AC-Critical-Pairs*(r_1, r_2)
  End While
  Return R
End Procedure AC-Completion

Procedure **Try-To-Conflate-Pairs**
  While $CP \neq \phi$ do
    $(s = t) :=$"smallest" member of CP
    $CP := CP - \{(s = t)\}$
    If $s\downarrow^{R,F} \overset{E}{\neq} t\downarrow^{R,F}$
    Then $r :=$ *Form-Reduction*$(s\downarrow^{R,E}, t\downarrow^{R,E})$
      $R, RR :=$ *Add-Reduction*(r,R,RR)
      $R, RR :=$ *Simplify-R*(R,RR)
  End While
  Return R, RR
End Procedure Try-To-Conflate-Pairs

Procedure **AC-Critical-Pairs** $(\lambda_1 \rightarrow \rho_1,\ \lambda_2 \rightarrow \rho_2)$
  Return
    $\{ < \rho_1\sigma, \rho_2\sigma > \mid \sigma \in AC\text{-}Unify(\lambda_1, \lambda_2)\} \cup$
    $\{ < \rho_1\sigma, (\lambda_1[m \leftarrow \rho_2])\sigma > \mid m \in Sdom(\lambda_1),\ m \neq \varepsilon,$
     and $\sigma \in AC\text{-}Unify(\lambda_1/m, \lambda_2)\} \cup$
    $\{ < (\lambda_2[m \leftarrow \rho_1])\sigma, \rho_2\sigma > \mid m \in Sdom(\lambda_2),\ m \neq \varepsilon,$
     and $\sigma \in AC\text{-}Unify(\lambda_1, \lambda_2/m)\}$
End Procedure AC-Critical-Pairs

Figure 11A. Peterson-Stickel AC-Completion Procedure - Part 1

3. Jouannaud-Kirchner E-Completion Procedure.

Jouannaud and Kirchner [*JK*86] generalized the Peterson-Stickel theory, lessening the requirements slightly and providing a different approach to the problems of E-compatibility and extensions. Maintaining a high degree of generality toward the implemented rewriting relation, they developed all of their theoretical results using a rewriting relation $\overset{R^E}{\rightarrow}$ which is free to be any rewriting relation satisfying the

```
Procedure Form-Reduction (s, t)
   Case W(s) > W(t):  Return s → t
        W(t) > W(s):  Return t → s
        Otherwise     Halt with Failure
End Procedure Form-Reduction

Procedure Add-Reduction (r,R,RR)
   R := R ∪ {q | q ∈ {r}^ac_E}
   RR := RR ∪ {(q, q') | q ∈ {r}^ac_E and q' ∈ R}
   Return R, RR
End Procedure Add-Reduction

Procedure Simplify-R
   For each r ∈ R do
      If r↓^(R−{r}),E ≠_E r
      Then RR := RR − {(q,q') | q ∈ {r}^ac_E and q' ∈ R}
           R := R − {q | q ∈ {r}^ac_E}
           If, for r↓^(R−{r}),E ≡ λ → ρ, λ ≠_E ρ
           Then r := Form-Reduction(λ, ρ)
                R, RR := Add-Reduction(r,R,RR)
                R, RR := Simplify-R(R,RR)
   Return R, RR
End Procedure Simplify-R
```

Figure 11B. Peterson-Stickel AC-Completion Procedure - Part 2

inequality $\overset{R}{\rightarrow} \subseteq \overset{R^E}{\rightarrow} \subseteq \overset{R/E}{\rightarrow}$. We now present a brief summary of their theory, highlighting the points where their theory differs from that presented previously.

a. Confluence and Coherence.

The properties of confluence and local confluence are formulated in terms of $\overset{R^E}{\rightarrow}$ as follows. A pair (t_1, t_2) is *R^E-confluent modulo E*, denoted $t_1 \downarrow t_2$ iff there exist terms t'_1 and t'_2 such that $t_1 \overset{R^E}{\underset{*}{\rightarrow}} t'_1$, $t_2 \overset{R^E}{\underset{*}{\rightarrow}} t'_2$, and $t'_1 \overset{E}{=} t'_2$. *R^E is confluent modulo E* iff for all terms t, t_1, and t_2, $t \overset{R^E}{\underset{*}{\rightarrow}} t_1$, and $t \overset{R^E}{\underset{*}{\rightarrow}} t_2 \Rightarrow t_1 \downarrow t_2$. *$R^E$ is locally confluent modulo E with R* iff for all terms t, t_1, and t_2, $t \overset{R^E}{\rightarrow} t_1$, and $t \overset{R}{\rightarrow} t_2 \Rightarrow t_1 \downarrow t_2$.

In place of the E-compatibility property of Peterson and Stickel, Jouannaud and Kirchner introduce the notion of *coherence*. Confluence and coherence are two instances of the same general concept: that a term may be mapped into two different forms (possibly by two different relations) and that these two forms may be brought back together by another relation. They define coherence and local coherence formally as follows: R^E *is coherent modulo* E iff for all terms t, t_1, and t_2, $t \overset{R^E}{\underset{*}{\rightarrow}} t_1$, and $t \overset{E}{=} t_2 \Rightarrow t_1 \downarrow t_2$. R^E *is locally coherent modulo* E iff for all terms t, t_1, and t_2, $t \overset{R^E}{\rightarrow} t_1$, and $t \overset{E}{=} t_2 \Rightarrow t_1 \downarrow t_2$. Thus confluence addresses the case where a term is pulled apart via two applications of $\overset{R^E}{\underset{*}{\rightarrow}}$ and coherence addresses the case where a term is pulled apart via $\overset{R^E}{\underset{*}{\rightarrow}}$ and E-equality. This comparison is illustrated in Figure 12.

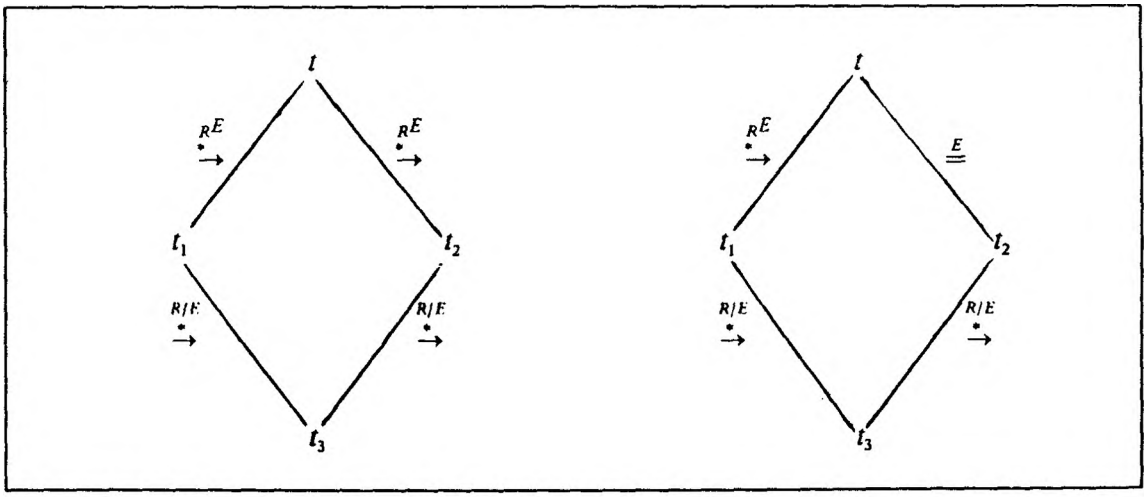

Figure 12. Confluence versus Coherence

The E-critical pairs defined in our discussion of the Peterson-Stickel E-completion procedure must now be further distinguished as *confluence critical pairs* to distinguish them from *coherence critical pairs*. *Coherence critical pairs* are computed in exactly the same manner as confluence critical pairs except that a reduction $\lambda \rightarrow \rho$ and an equation $l = r$ play the roles of $\lambda_1 \rightarrow \rho_1$ and $\lambda_2 \rightarrow \rho_2$. Following much the same pattern as the original Knuth-Bendix theory, the local confluence and

local coherence properties will be reduced to the confluence of their critical pairs, respectively. The notions of confluence and coherence, playing similar roles, provide the core of the Jouannaud-Kirchner E-completion theory.

b. Church-Rosser Properties.

R is defined to be *Church-Rosser modulo* E iff for all terms t_1 and t_2, $t_1 \overset{R \cup E}{=} t_2 \Rightarrow$ there exists a term t_3 such that $t_1 \overset{R/E}{\bullet\!\!\rightarrow} t_3$ and $t_2 \overset{R/E}{\bullet\!\!\rightarrow} t_3$. The following theorem highlights the roles of confluence and coherence in achieving this property.

E-Church-Rosser Theorem: [*JK*86]

If the rewriting relation $\overset{R/E}{\rightarrow}$ *satisfies the finite termination property, then the following properties are equivalent:*

(1) R *is* R^E *Church-Rosser modulo* E.

(2) R^E *is confluent modulo* E *and* R^E *is coherent modulo* E.

(3) R^E *is locally confluent modulo* E *and* R^E *is locally coherent modulo* E.

(4) for all terms t_1 *and* t_2, $t_1 \overset{R \cup E}{=} t_2$ *iff* $t_1{\downarrow}^{R^E} \overset{E}{=} t_2{\downarrow}^{R^E}$.

We now give the main theorem of the Jouannaud-Kirchner paper, in a slightly simplified form. Their theory allows, for purposes of efficiency, the separation of the set R into two sets, L and N, such that all of the reductions in L satisfy the requirement that no variable appears more than once in the left side of a reduction. It is permissible under their theory, however, to have such rules in N. We will simplify matters by leaving all of the reductions in R for our purposes, treating R as their N.

E-Church-Rosser Decidability Theorem: [*JK*86]

Assume an equational theory E *such that a finite and complete unification algorithm exists for* E *and* E*-congruence classes are finite. Let* R *be a set of reductions such that* $\overset{R/E}{\rightarrow}$ *satisfies the finite termination property.* R *is* R^E-Church-Rosser *modulo* E *iff:*

(1) all confluence critical pairs $< c_1, c_2 >$ *in* $\cup_{(p,q)\in R\times R} E\text{-}Critical\text{-}Pairs(p, q)$ *are* R/E *confluent modulo* E.

(2) for any coherence critical pair $< c_1, c_2 >$ *in* $\cup_{(r,e)\in R\times E} E\text{-}Critical\text{-}Pairs(r,e)$ *there exists a term* c'_2 *such that* $c_2 \overset{RE}{\rightarrow} c'_2$ *and* $< c_1, c'_2 >$ *is* R/E *confluent modulo* E.

The proofs of these last two theorems are based on multiset induction, a special case of the noetherian induction technique used in Huet's proof of the original Knuth-Bendix theory.

Note that the E-compatibility property of Peterson and Stickel has been replaced by the confluence of coherence critical pairs and that the linearity and non-erasing requirements for E have been replaced by the single requirement that E generate finite congruence classes.

c. Generalized Extensions.

In the Jouannaud-Kirchner theory, the concept of an extended reduction is improved in two ways over extensions as presented in the Peterson-Stickel theory. First, rather than systematically adding extensions for every reduction whose left side is rooted with an AC operator, Jouannaud and Kirchner examine the coherence critical pairs of a reduction with the equations of E. Only when there is a pair which will not conflate do they add an extended reduction to R. They call this the concept of *dynamic extensions*. Secondly, they generalize the concept of AC extensions such that they can generate an extension for any reduction, no matter what the equational theory may be. Suppose that the reduction $\lambda \rightarrow \rho$ and the equation $l = r$ produce a coherence critical pair which does not conflate when l/m and λ are E-unified. They show that the new reduction $l[m \leftarrow \lambda] \rightarrow l[m \leftarrow \rho]$ is an extension which will conflate the troublesome pair. This does not depend on $l = r$ being part of an AC equational theory, as in the Peterson-Stickel theory.

4. Kaplan-Remy Completion for Conditional Reductions.

Moving in a slightly different direction, we now present a brief look at the theory of completion for a set of conditional reductions. We address this because the completion process presented in later chapters turns out to be E-completion of conditional reductions. Although the conditions which arise in this research are somewhat specialized to our problem, they do maintain enough generality as to be similar to the conditions of other researchers.

Kaplan and Remy [*KR*87] address the standard completion problem, without respect to an equational theory, for conditional reductions of the form

$$\text{If } u_1 = v_1 \text{ and } u_2 = v_2 \text{ and } \ldots \text{ and } u_n = v_n \text{ Then } \lambda \rightarrow \rho.$$

In order to apply one of these reductions, a term matching substitution σ is found between λ and the term to be reduced. If this match is successful, then σ is applied to the condition and a check is made to see if the condition holds. In their theory this evaluation involves applying the reductions recursively to the terms of the condition; in our case it will not. Aside from this difference, our conditions will be used in the same manner.

Kaplan and Remy define a *steady conditional rewriting system* to be one in which the variables in the condition and in ρ also all appear in λ. A *contextual critical pair* is defined to be a critical pair of the same form as the Knuth-Bendix critical pair, with an associated *critical context* which is the result of applying the unifier used to produce the critical pair to the conjunction of the conditions from each of the reductions involved in the critical pair. A contextual critical pair is then said to be *feasible* iff the critical context holds. This leads to the following theorem.

Church-Rosser Theorem for Conditional Rewriting Systems: [*KR*87]

Given a steady conditional rewriting system R, $\xrightarrow{R}$ *is locally confluent, and thus confluent, iff for every feasible contextual critical pair* $< c_1, c_2 >$ *with critical context C,* $c_1\sigma\downarrow^R = c_2\sigma\downarrow^R$ *for all substitutions* σ *which cause the critical context to hold.*

Although the form and manner of application of our conditional reductions will be slightly different, we will build on the concept of subjecting critical pairs to the conditions of both involved reductions. We will also apply this concept to coherence critical pairs as we blend the theory of E-completion with the theory of completion for conditional reductions.

IV. TERMINATION VIA CONDITIONAL REDUCTIONS

A. INTRODUCTION

Much of the work in term rewriting relative to an equational theory, E, has involved the use of the $\overset{R/E}{\rightarrow}$ and $\overset{R,E}{\rightarrow}$ rewriting relations. In general, $\overset{R/E}{\rightarrow}$ has been used to develop theoretical results and some form of $\overset{R,E}{\rightarrow}$ has been used in computer programs which implement those theories. Both of these are rewriting relations between elements of congruence classes generated by the equational theory and have been shown to terminate when the equational theory generates finite congruence classes, such as those generated by an AC equational theory. When we move to equational theories which generate infinite congruence classes, such as those generated by an ACI equational theory, however, we may lose the termination property for both of these rewriting relations.

The fact that we may lose termination when rewriting relative to an ACI equational theory is significant for two reasons. In the first place, most theoretical results related to establishing that a set of reductions is complete relative to an equational theory depend on the termination of $\overset{R/E}{\rightarrow}$. Jouannaud and Kirchner [$JK86$] develop the theory of completing a set of reductions relative to an equational theory provided the equational theory generates finite congruence classes and $\overset{R/E}{\rightarrow}$ terminates. In the same work it is also noted that a significant open problem in this area is the generalization of the completion theory to handle equational theories which generate infinite congruence classes. Bachmair and Plaisted [$BP87$] generalize the previous work, removing the requirement of finite congruence classes, but still requiring the termination of $\overset{R/E}{\rightarrow}$. This generalization does not help, however, if the termination of $\overset{R/E}{\rightarrow}$ is lost for equational theories which generate infinite congruence classes. In what follows we will demonstrate that this is often the case. The second

reason for the significance of the loss of termination is quite simple. Aside from any theoretical results relating to completeness, reductions can still be very useful for simplifying expressions. This usefulness is severely limited, however, if the implemented rewriting relation must deal with the possibility of infinite chains.

Recall from Chapter 2 that $\overset{R,E}{\rightarrow}$ and $\overset{R/E}{\rightarrow}$ are defined such that

$$t_1 \overset{R,E}{\rightarrow} t_2 \Rightarrow t_1 \underset{\leq m}{\overset{E}{=}} t'_1 \overset{R}{\rightarrow} t_2$$

and

$$t_1 \overset{R/E}{\rightarrow} t_2 \quad \textit{iff} \quad t_1 \overset{E}{=} t'_1 \overset{R}{\rightarrow} t'_2 \overset{E}{=} t_2 .$$

Note that $\overset{R,E}{\rightarrow} \subseteq \overset{R/E}{\rightarrow}$, thus if $\overset{R,E}{\rightarrow}$ contains infinite chains $\overset{R/E}{\rightarrow}$ will contain them also. The following example demonstrates that $\overset{R,E}{\rightarrow}$ and $\overset{R/E}{\rightarrow}$ termination can, in fact, both be lost when rewriting relative to an ACI equational theory.

Let R contain the reduction $-(x+y) \rightarrow (-x)+(-y)$ and let E be the ACI equational theory for $+$. Then the term $(-a)$ can be rewritten as

$$(-a) \overset{E}{=} -(a+0) \overset{R}{\rightarrow} (-a)+(-0) \overset{E}{=} -(a+0)+(-0) \overset{R}{\rightarrow} ((-a)+(-0))+(-0) \overset{E}{=} \ldots$$

When rewritten as an $\overset{R,E}{\rightarrow}$ chain we have

$$(-a) \overset{R,E}{\rightarrow} (-a)+(-0) \overset{R,E}{\rightarrow} ((-a)+(-0))+(-0) \overset{R,E}{\rightarrow} \ldots$$

Clearly both $\overset{R,E}{\rightarrow}$ and $\overset{R/E}{\rightarrow}$ contain infinite chains in this example. It is very easy to find many other similar examples where termination is lost for ACI equational theories and for other equational theories which generate infinite congruence classes.

In order to develop any theory for rewriting relative to an ACI equational theory, we must first develop a rewriting relation which is provably terminating. In the following we develop a generalization of the $\overset{R/E}{\rightarrow}$ rewriting relation for rewriting

relative to ACI equational theories and establish the criteria under which its termination is guaranteed.

B. PRELIMINARIES

Before we develop the termination criteria for $\overset{R/E}{\rightarrow}$, we must first introduce two other concepts which will be used in the proofs of termination. These are the concepts of *core elements* and *properties of weighting functions.*

1. Core Elements.

We now define the notion of a core element of a congruence class generated by an ACI equational theory. We will say that a term t is a *core element* of $[t]_{ACI}$ if t is in normal form with respect to the rewriting relation $\overset{I,AC}{\rightarrow}$, where I is the set of reductions of the form $x + 0 \rightarrow x$ and AC is the set of associative and commutative laws for each ACI operator, $+$, in the equational theory. The rewriting relation used here is precisely the same as $\overset{R,E}{\rightarrow}$ with I playing the role of R and AC playing the role of E. We will write $t\downarrow^I$ to mean the normal form of t with respect to $\overset{I,AC}{\rightarrow}$. Note that I is by itself a complete set of reductions with respect to AC, thus all core elements of $[t]_{ACI}$ are AC-equal to each other. Clearly this means that there are a finite number of terms in the core for any congruence class generated by an ACI theory. Furthermore, given any term of finite size, we can easily find the associated core element.

For example, consider the ACI congruence class which contains the terms $a + b$, $(a + b) + 0$, $(a + 0) + b$, $b + (0 + a)$, $(a + 0) + (0 + b)$, $(0 + 0) + (b + a)$, $(((a + b) + 0) + 0)$, ... The core for this congruence class contains only the two terms $a + b$ and $b + a$.

2. Weighting Function Properties.

As is usually the case, our proof of termination for the $\overset{R/E}{\rightarrow}$ rewriting relation will be based on the use of a weighting function, W, such that $W(t)$ gives the weight of any term t. We will depend on the following six properties for W:

$W1$: $\forall t\ W(t) > 0$

$W2$: i is an identity for an ACI operator in $E \Rightarrow \forall t\ W(i) \leq W(t)$

$W3$: $s \overset{AC}{=} t \Rightarrow W(s) = W(t)$

$W4$: $W(s) > W(t) \Rightarrow W(T[m \leftarrow s]) > W(T[m \leftarrow t])$

$W5$: $W(s) > W(t)$ and θ is any substitution $\Rightarrow W(s\theta) > W(t\theta)$

$W6$: $t/m = s$, for some $m \in Dom(t) \Rightarrow W(s) \leq W(t)$

These properties have been shown for a number of weighting functions. Since weighting functions are usually dependent on the actual operators allowable in s and t, we will assume that such a W exists. The required properties can then be demonstrated when the sets R and E have been given, making known the allowable operators for s and t. For problems which involve the ACI operators $+$ and $*$ and the unary operator $-$, the complexity measures of Lankford [$La79$] have been shown to meet the required properties.

Another possible approach to this problem would have been to develop a new weighting function which handles some of the problems which we encounter when dealing with infinite congruence classes. For instance, we could have attempted to develop a weighting function which assigns the same weight to all members of an ACI congruence class. In doing this, however, we would lose property $W5$, which seems to be more useful than the suggested property. Our present approach, therefore, is to work with weighting functions similar to those which have already been developed by those working with finite congruence classes under AC theories.

C. R/E TERMINATION

In this section we establish sufficient conditions for the termination of $\overset{R/E}{\rightarrow}$. The basic approach is to demonstrate criteria under which the weight of a term strictly decreases on every $\overset{R/E}{\rightarrow}$ step. We first present and prove a theorem which indicates these requirements. This result is then used to redefine the notion of $\overset{R/E}{\rightarrow}$ rewriting.

1. Termination Theorem.

In order to accomplish our goal in this section we begin by proving a group of lemmas which allow us to reduce the problem to that of classifying the substitution involved in the rewriting. Necessary terminology and functions will be defined along the way.

The following property of ACI congruence classes was mentioned informally in our previous discussion of core elements. We state it more formally here for reference in a later proof.

Lemma 1: If $t \overset{ACI}{=} s$, $t{\downarrow^I} \overset{AC}{=} s{\downarrow^I}$.

Proof: This is a direct consequence of the definition of $\downarrow^I$ and the fact that I is by itself a complete set of reductions with respect to AC. □

We now show that coring a term can never increase its weight.

Lemma 2: For every term, t, $W(t{\downarrow^I}) \leq W(t)$.

Proof: It will suffice to show that if s is obtained from t by one application of an identity law, then $W(s) \leq W(t)$. We assume without loss of generality that the identity law is $x + 0 \rightarrow 0$, for some ACI operator $+$. There must be some $m \in Dom(t)$ such that $t/m = u + 0$ for some term u, and $s = t[m \leftarrow u]$. By W6, $W(u) \leq W(u + 0)$, and by W4, $W(s) = W(t[m \leftarrow u]) \leq W(t[m \leftarrow (u + 0)]) = W(t)$. □

The next lemma makes it clear that we can preserve ACI-equality when we substitute equals for equals on both sides of the equality, provided that the subterm being replaced is in the same context in each term, relative to the ACI theory. This contextual requirement is assured by the added condition that the subterm occurs exactly once in each side. It is easy to see that the lemma is not true without this contextual requirement.

Lemma 3: Given terms t and t', a constant c, and positions $x \in dom(t)$ and $x' \in dom(t')$ such that $t[x \leftarrow c] \overset{ACI}{=} t'[x' \leftarrow c]$, $c \overset{ACI}{\neq} ident(\alpha)$ for any ACI operator α in E, and c occurs in neither t nor t', then for any term s, $t[x \leftarrow s] \overset{ACI}{=} t'[x' \leftarrow s]$.

Proof: Since we are given that $t[x \leftarrow c] \overset{ACI}{=} t'[x' \leftarrow c]$, this means that there exists a sequence of terms $t[x \leftarrow c] \equiv t_1 \overset{ACI}{\underset{1}{=}} t_2 \overset{ACI}{\underset{1}{=}} \ldots \overset{ACI}{\underset{1}{=}} t_n \equiv t'[x' \leftarrow c]$, where $\overset{ACI}{\underset{1}{=}}$ is used to mean a single application of one of the ACI equations. Since none of these equations can eliminate or duplicate c it follows that there is exactly one occurrence of c in each t_i. This means that a corresponding sequence of $\overset{ACI}{\underset{1}{=}}$ steps with each c replaced by s can be used to demonstrate that $t[x \leftarrow s] \overset{ACI}{=} t'[x' \leftarrow s]$. □

We now establish the existence of a core term which is similar enough in structure to a given term that we can replace a subterm in each with ACI-equal terms and preserve ACI-equality. This lemma will provide the backbone for the proof of our main theorem in this section.

Lemma 4: Given a term, t, and a position, $x \in dom(t)$, then there exists a core term, t', and a position, $x' \in dom(t')$, such that for any term s, $t[x \leftarrow s] \overset{ACI}{=} t'[x' \leftarrow s{\downarrow}^t]$.

Proof: Let $t' = (t[x \leftarrow c]){\downarrow}^t$ where c is a special constant not previously appearing in t and $c \overset{ACI}{\neq} ident(\alpha)$ for any ACI operator α in E. The special constant c will serve as a marker to mark position x in t and allow the determination of the corresponding position in t' after the coring process has taken place. Clearly the rewriting relation $\overset{I,AC}{\rightarrow}$ can move the position of c during the coring process, however, it can neither eliminate nor duplicate c since the AC equations can only serve to permute terms and

the I reduction can only eliminate identities, which c is not. Thus there must be one and only one position $x' \in dom(t')$ such that $t'/x' = c$. We know that $t[x \leftarrow c] \overset{ACI}{=} (t[x \leftarrow c])\downarrow^I$ since $\downarrow^I$ preserves ACI-equality. But $(t[x \leftarrow c])\downarrow^I = t'$ by definition, and since $t'/x' = c$, we now have $t[x \leftarrow c] \overset{ACI}{=} t'[x' \leftarrow c]$. Since c occurs exactly once on each side of this equation, we can apply Lemma 3 and substitute any term s for the marker giving $t[x \leftarrow s] \overset{ACI}{=} t'[x' \leftarrow s]$. Finally, we can core s on one side since coring preserves ACI-equality and we have $t[x \leftarrow s] \overset{ACI}{=} t'[x' \leftarrow s\downarrow^I]$. □

It is important to note that the term s can be changed arbitrarily after t' and x' have been found. This will allow us to find t' for a given t and then change the substituted subterm without having to find another $t - t'$ pair.

Our next lemma will be used later to establish that, under the conditions which we will assume, $\overset{R/E}{\rightarrow}$ cannot replace a subterm which is E-equal to an identity.

Lemma 5: If $W(t\downarrow^I) > W(s\downarrow^I)$ then $t\downarrow^I \overset{ACI}{\neq} ident(\alpha)$ for any ACI operator α in E.

Proof: Assume $t\downarrow^I \overset{ACI}{=} ident(\alpha)$. This implies that $t\downarrow^I \overset{AC}{=} ident(\alpha)$ by Lemma 1 since both are core terms. Then by $W3$ we have $W(ident(\alpha)) = W(t\downarrow^I)$. But this together with the given hypothesis allows us to conclude that $W(ident(\alpha)) > W(s\downarrow^I)$, which contradicts $W2$. Thus the assumption that $t\downarrow^I \overset{ACI}{=} ident(\alpha)$ must be false. □

The following lemma presents another result which will be needed in the proof of our main theorem. This shows that coring the subterm inserted into a cored term is equivalent to coring the resulting term, provided that the inserted subterm does not collapse down to an identity.

Lemma 6: If $y \in dom(t\downarrow^I)$ and $s\downarrow^I \overset{ACI}{\neq} ident(\alpha)$ for any ACI operator α in E, then $(t\downarrow^I[y \leftarrow s])\downarrow^I = t\downarrow^I[y \leftarrow s\downarrow^I]$.

Proof: Clearly $t\downarrow^I[y \leftarrow s\downarrow^I]$ is in normal form with respect to $\overset{I,AC}{\rightarrow}$ unless $s\downarrow^I = ident(\alpha)$ for some ACI operator α in $t\downarrow^I$ which has $s\downarrow^I$ in its scope. Since we are given that $s\downarrow^I \neq ident(\alpha)$ for any ACI operator α in E, this cannot be the case. Thus

$t\downarrow^I[y \leftarrow s\downarrow^I] = (t\downarrow^I[y \leftarrow s\downarrow^I])\downarrow^I$, which is equal to $(t\downarrow^I[y \leftarrow s])\downarrow^I$ by the definition of $\downarrow^I$. □

Given a substitution $\sigma = \{x_1 \leftarrow t_1, x_2 \leftarrow t_2, \ldots, x_n \leftarrow t_n\}$ and a term t, we can split σ into two disjoint portions by defining functions Σ_1 and Σ_2 as follows:

$$\Sigma_1(\sigma, t) = \{(x_i \leftarrow t_i) \mid (x_i \leftarrow t_i) \in \sigma \text{ and } x_i \text{ is in the scope of an ACI operator, } \alpha, \text{ in } t \text{ and } t_i \stackrel{E}{=} \mathit{Ident}(\alpha), \text{ the identity for } \alpha.\}$$

$$\Sigma_2(\sigma, t) = \sigma - \Sigma_1(\sigma, t)$$

Clearly, if $\sigma_1 = \Sigma_1(\sigma, t)$ and $\sigma_2 = \Sigma_2(\sigma, t)$ then $\sigma = \sigma_1 \cup \sigma_2 = \sigma_1\sigma_2$. Our main theorem will show that the termination of $\stackrel{R/E}{\rightarrow}$ is dependent only on the Σ_1 portion of the term matching substitution which is used to apply each reduction. In order to show that the Σ_1 portion of a substitution plays the vital role in this process, we will first consider the role of the Σ_2 portion.

Lemma 7: Consider a substitution σ and a term t. Let $\sigma_2 = \Sigma_2(\sigma, t)$ and define a *cored substitution*, $\sigma\downarrow^I$, by

$$\sigma\downarrow^I = \{(x_i \leftarrow t_i\downarrow^I) \mid (x_i \leftarrow t_i) \in \sigma\},$$

then $(t\sigma_2)\downarrow^I = (t\downarrow^I)\ \sigma_2\downarrow^I$.

Proof: The only way these terms can differ is if $\sigma_2\downarrow^I$ can introduce a context for the application of $\downarrow^I$ into $t\downarrow^I$ causing $(t\downarrow^I)\sigma_2\downarrow^I$ not to be a core term while $(t\sigma_2)\downarrow^I$ is clearly a core term. This cannot happen, however, because if the context for $\downarrow^I$ had been in t_i it has already been eliminated and if $t_i \stackrel{ACI}{=} \mathit{Ident}(\alpha)$ where x_i is in the scope of an ACI operator, α, in t, then $(x_i \leftarrow t_i) \notin \sigma_2\downarrow^I$ by definition of Σ_2. □

As the final piece which we will need in order to prove our main theorem for this section, we now define the *restricted substitution set*, $\Theta(\lambda \rightarrow \rho)$, as follows:

$$\Theta(\lambda \rightarrow \rho) = \{\sigma \mid \sigma = \{x_1 \leftarrow \mathit{Ident}(\alpha_1), \ldots x_n \leftarrow \mathit{Ident}(\alpha_n)\}, \text{ where}$$

$n \geq 0$,

each α_i is an ACI operator,

each x_i is a variable in λ which is in the scope of α_i, and

$W((\lambda\sigma)\downarrow^I) \leq W((\rho\sigma)\downarrow^I)\}$.

We now state and prove our main result: that $\overset{R/E}{\rightarrow}$ must terminate when the conditions for each rule, represented by $\Theta(\lambda \rightarrow \rho)$, are enforced.

Termination Theorem:

If the reduction $t \underset{\lambda,\rho,\sigma}{\overset{R/E}{\rightarrow}} s$ is allowed to take place only when $\sigma \notin \iota_E\Theta(\lambda \rightarrow \rho)$, then the rewriting relation $\overset{R/E}{\rightarrow}$ must terminate.

Proof: It will be sufficient to show that, under the given conditions, $W(t\downarrow^I) > W(s\downarrow^I)$. If $t_1 \overset{R/E}{\rightarrow} t_2 \overset{R/E}{\rightarrow} \ldots$ is an infinite sequence of $\overset{R/E}{\rightarrow}$ reductions, then $t_1\downarrow^I$, $t_2\downarrow^I$, ... is an infinite sequence of terms whose weights get strictly smaller, but this is impossible by W1. We proceed as follows: By the definition of $t \underset{\lambda,\rho,\sigma}{\overset{R/E}{\rightarrow}} s$, there exist terms t' and s' such that

$$t \overset{E}{=} t' \underset{\lambda,\rho,\sigma}{\overset{R}{\rightarrow}} s' \overset{E}{=} s.$$

Since $t \overset{E}{=} t\downarrow^I$, it follows that

$$t\downarrow^I \overset{E}{=} t' \underset{\lambda,\rho,\sigma}{\overset{R}{\rightarrow}} s' \overset{E}{=} s\downarrow^I.$$

By the definition of $\underset{\lambda,\rho,\sigma}{\overset{R}{\rightarrow}}$, there exists $m \in Dom(t')$ such that $t'/m = \lambda\sigma$ and $\sigma' = t'[m \leftarrow \rho\sigma]$. By Lemma 4 there exists a core term t'' and a position $m' \in Dom(t'')$ such that for every term u,

$$(1) \quad t'[m \leftarrow u] \overset{E}{=} t''[m' \leftarrow u\downarrow^I].$$

Let $\sigma_1 = \Sigma_1(\sigma, \lambda)$ and $\sigma_2 = \Sigma_2(\sigma, \lambda)$. From the definitions of Σ_1 and Σ_2 it is clear that $\sigma = \sigma_1\sigma_2$. From the definition of E-equality for substitutions, it follows that $\sigma \overset{E}{=} \sigma_1\downarrow^I\sigma_2$. Since for each $(x_i \leftarrow t_i) \in \sigma_1$ the definition of Σ_1 gives that $t_i \overset{E}{=} Ident(\alpha_i)$, it follows that $t_i\downarrow^I = Ident(\alpha_i)$. From the given condition, $\sigma \notin \iota_E\Theta(\lambda \rightarrow \rho)$, we see that $\sigma_1\downarrow^I \notin \Theta(\lambda \rightarrow \rho)$, giving

$$
\begin{aligned}
W((\lambda\sigma_1)\downarrow') &= W((\lambda\sigma_1\downarrow')\downarrow') && \text{by Lemma 1 and W3} \\
&> W((\rho\sigma_1\downarrow')\downarrow') && \text{by the definition of } \Theta(\lambda \to \rho) \\
&= W((\rho\sigma_1)\downarrow') && \text{by Lemma 1 and W3,}
\end{aligned}
$$

and

$$
\begin{aligned}
W((\lambda\sigma)\downarrow') &= W((\lambda\sigma_1)\downarrow'\sigma_2\downarrow') && \text{by Lemma 7} \\
&> W((\rho\sigma_1)\downarrow'\sigma_2\downarrow') && \text{by W5} \\
&\geq W(((\rho\sigma_1)\downarrow'\sigma_2\downarrow')\downarrow') && \text{by Lemma 2} \\
&= W((\rho\sigma_1\sigma_2)\downarrow') && \text{by Lemma 1 and W3} \\
&= W((\rho\sigma)\downarrow') && \text{by definitions of } \Sigma_1 \text{ and } \Sigma_2.
\end{aligned}
$$

Now Lemma 5 assures us that $(\lambda\sigma)\downarrow'$ cannot be an identity. We conclude that

$$
\begin{aligned}
W(t\downarrow') &= W(t'\downarrow') && \text{by Lemma 1 and W3} \\
&= W((t'[m \leftarrow \lambda\sigma])\downarrow') && \text{since } t'/m = \lambda\sigma \\
&= W((t''[m' \leftarrow (\lambda\sigma)\downarrow'])\downarrow') && \text{by (1) above} \\
&= W(t''[m' \leftarrow (\lambda\sigma)\downarrow']) && \text{by Lemma 6} \\
&> W(t''[m \leftarrow '(\rho\sigma)\downarrow']) && \text{by W4} \\
&\geq W((t''[m' \leftarrow (\rho\sigma)\downarrow'])\downarrow') && \text{by Lemma 2} \\
&= W((t'[m \leftarrow \rho\sigma])\downarrow') && \text{by (1), Lemma 1 and W3} \\
&= W(s'\downarrow') && \text{since } s' = t'[m \leftarrow \rho\sigma] \\
&= W(s\downarrow') && \text{by Lemma 1 and W3.} \qquad \square
\end{aligned}
$$

2. A Generalization of R/E.

We now propose to redefine the notion of $\overset{R/E}{\rightarrow}$ rewriting as follows:

$$
t_1 \underset{\lambda,\rho,\sigma}{\overset{R/E}{\rightarrow}} t_2 \quad \textit{iff} \quad t_1 \overset{E}{=} t'_1 \underset{\lambda,\rho,\sigma}{\overset{R}{\rightarrow}} t'_2 \overset{E}{=} t_2 \text{ and } \sigma \notin \iota_E\Theta(\lambda \to \rho)
$$

Note that the conditional version of $\overset{R/E}{\rightarrow}$ can be thought of as a generalization of the normal definition of $\overset{R/E}{\rightarrow}$. All that is required is to have "empty" conditions on reductions of the normal $\overset{R/E}{\rightarrow}$ variety. When viewed as such, any theory developed

around conditional reductions subsumes a similar theory developed around the usual unconditional reductions. Hereafter we will use $\overset{R/E}{\rightarrow}$ to refer to this generalization. Peterson et al. [*PB87*] present a procedure for testing the completeness of a set of reductions relative to an ACI equational theory, based on the conditional version of the $\overset{R/E}{\rightarrow}$ rewriting relation presented above. It was assumed in that study that $\overset{R/E}{\rightarrow}$ did terminate, subject to the conditions, and the main proofs were based on that assumption. Our termination result thus collaborates that assumption. In the following chapter we will develop a procedure for completing a set of reductions relative to an ACI equational theory, using the generalized conditional rewriting relation.

D. APPLYING THE TERMINATION THEOREM

1. Calculating Conditions.

We new describe a simple procedure for calculating the conditions which are needed for each reduction in order to satisfy the termination property. Recall from the previous section that the conditions for each reduction are represented by the restricted substitution set $\Theta(\lambda \rightarrow \rho)$ which is defined by:

$$\begin{aligned}\Theta(\lambda \rightarrow \rho) = \{\sigma \mid \sigma = \{x_1 \leftarrow \mathit{Ident}(\alpha_1), \ldots x_n \leftarrow \mathit{Ident}(\alpha_n)\},\ &\text{where}\\ &n \geq 0,\\ &\text{each } \alpha_i \text{ is an ACI operator,}\\ &\text{each } x_i \text{ is a variable in } \lambda \text{ which is in the scope of } \alpha_i, \text{ and}\\ &W((\lambda\sigma)\downarrow') \leq W((\rho\sigma)\downarrow')\}.\end{aligned}$$

We begin by finding the set $I(\lambda)$, where $I(t)$ is given by

$$I(t) = \{(x \leftarrow \mathit{Ident}(\alpha)) \mid x \text{ is a variable in } t \text{ in the scope of an ACI operator, } \alpha\}.$$

The set $I(\lambda)$ then forms the basis of identity substitution pairs from which all possible members of $\Theta(\lambda \rightarrow \rho)$ will be generated. We then generate potential substitutions, $P(\lambda)$, where $P(t)$ is given by

$$P(t) = \{y \mid y \in 2^{I(\lambda)} \text{ and } y \text{ is a valid substitution}\}.$$

Clearly, the powerset, $2^{I(\lambda)}$, generates all possible combinations of identity substitution pairs. We must discard any substitution which assigns more than one identity to the same variable, however, because these are not valid substitutions. Finally, we test each member, σ, of $P(\lambda)$ to see whether or not $W((\lambda\sigma)\downarrow^{\eta}) \leq W((\rho\sigma)\downarrow^{\eta})$. If the test succeeds we place σ in $\Theta(\lambda \rightarrow \rho)$, otherwise we do not.

Example 1: The following example illustrates how the preceding procedure is applied to a set of reductions to ensure $\overset{R/E}{\rightarrow}$ termination when E is an ACI equational theory. Consider the following set of reductions where + is an ACI operator and − has none of the ACI properties:

R1: $x + (-x) \rightarrow 0$

R2: $-(-x) \rightarrow x$

R3: $-(x+y) \rightarrow (-x) + (-y)$

For each of the examples which we present in this section we will use the weighting function $W(t)$ which is defined as follows:

$W(constant) = 2$ $\qquad$ $W(variable) = 2$

$W(x*y) = W(x)*W(y)$ $\qquad$ $W(x+y) = W(x) + W(y) + 5$

$W((-x)) = 2 + 2*W(x)$

For R1 the only variable in the scope of an ACI operator is x. The corresponding ACI operator is + and the corresponding identity is 0. This gives $I(\lambda) = \{x \leftarrow 0\}$ and $P(\lambda) = \{\phi, \{x \leftarrow 0\}\}$. Using these substitutions for σ, we find that $\forall\sigma$

$W(\lambda\sigma\downarrow^{\prime}) > W(\rho\sigma\downarrow^{\prime})$, thus no restrictions are needed for R1. R2 has no variables in the scope of ACI operators, giving $I(\lambda) = P(\lambda) = \Theta(\text{R2}) = \phi$. Thus R2 must only satisfy the property $W(\lambda) > W(\rho)$, which it does. R3 has variables x and y in the scope of the ACI operator + with the corresponding identity 0. This gives $I(\lambda) = \{x \leftarrow 0,\ y \leftarrow 0\}$, and $P(\lambda) = \{\ \phi,\ \{x \leftarrow 0\},\ \{y \leftarrow 0\},\ \{x \leftarrow 0,\ y \leftarrow 0\}\ \}$. Calling these substitutions σ_1, σ_2, σ_3, and σ_4, respectively, we find that $W(\lambda\sigma\downarrow^{\prime}) \leq W(\rho\sigma\downarrow^{\prime})$ for all substitutions $\sigma = \sigma_i$ except $\sigma = \sigma_1$. Thus Θ(R3) becomes $\{\sigma_2,\ \sigma_3,\ \sigma_4\}$. Since σ_4 is an instance of σ_2 and σ_3, however, any substitution which is an E-instance of σ_4 will also be an E-instance of σ_2 and σ_3. Because of this we will get the same result with Θ(R3) $= \{\sigma_2,\ \sigma_3\}$ as with Θ(R3) $= \{\sigma_2,\ \sigma_3,\ \sigma_4\}$. For the sake of simplicity we will use the more concise form. We now have the restrictions $\Theta(\text{R1}) = \phi$, $\Theta(\text{R2}) = \phi$, and $\Theta(\text{R3}) = \{\ \{x \leftarrow 0\},\ \{y \leftarrow 0\}\ \}$. Equivalently, the set of reductions which guarantees $\overset{R/E}{\rightarrow}$ termination can be represented as the set of conditional reductions given below:

R1: $\quad x + (-x) \rightarrow 0$

R2: $\quad -(-x) \rightarrow x$

R3: $\quad$ If $x \neq 0$ and $y \neq 0$ then $-(x+y) \rightarrow (-x) + (-y)$

This set has been shown to be a complete set of reductions for abelian groups relative to the ACI equational theory for +.

The preceding example suggests a better procedure for computing $\Theta(\lambda \rightarrow \rho)$. When λ contains at least one variable there will always be substitutions in $P(\lambda)$ which are instances of other substitutions in $P(\lambda)$ because the powerset of $I(\lambda)$ will contain members which are supersets of other members. For instance, as shown above, $P(x+y) = \{\phi, \{x \leftarrow 0\}, \{y \leftarrow 0\}, \{x \leftarrow 0, y \leftarrow 0\}\}$. Calling these substitutions σ_1, σ_2, σ_3, and σ_4, respectively, it is clear that σ_2, σ_3, and σ_4 are supersets of σ_1 making them instances of σ_1, and σ_4 is likewise an instance of both σ_2 and σ_3. This suggests that we generate and test the elements of the powerset from the smallest to the largest. If an

element p of $P(\lambda)$ is placed in $\Theta(\lambda \rightarrow \rho)$ then no larger element q of $P(\lambda)$ which is a superset of p need even be tested as to whether or not $W((\lambda q)\downarrow') \leq W((\rho q)\downarrow')$. The test would indicate that q should be added to $\Theta(\lambda \rightarrow \rho)$, but we know that we can leave it out. Because of the manner in which the substitutions are used, this clearly will not change the effect of $\Theta(\lambda \rightarrow \rho)$ but will speed up its calculation while automatically providing the restrictions in the most concise form. An interesting result of the process is that $\Theta(\lambda \rightarrow \rho) = \phi$ represents a reduction with no restrictions while $\Theta(\lambda \rightarrow \rho) = \{\phi\}$ represents a reduction which is always restricted, since every substitution is an instance of the empty substitution.

Example 2: In this example we will calculate restrictions using the procedure just described, so as to obtain minimal restrictions. Consider the following set of reductions:

R4: $x*(y+z) \rightarrow (x*y)+(x*z)$

R5: $x*0 \rightarrow 0$

R6: $x*(-y) \rightarrow -(x*y)$

For R4, $I(\lambda) = \{x \leftarrow 1,\ y \leftarrow 0,\ z \leftarrow 0\}$ and when we generate the powerset elements from the smallest to the largest we find that the singleton sets $\{x \leftarrow 1\}$, $\{y \leftarrow 0\}$, and $\{z \leftarrow 0\}$ are all added to $\Theta(\lambda \rightarrow \rho)$. No larger members need be tested as all larger members are supersets of at least one of these sets. For both R5 and R6 we find $I(\lambda) = \{x \leftarrow 1\}$, $P(\lambda) = \{\phi,\ \{x \leftarrow 1\}\}$, and $\Theta(\lambda \rightarrow \rho) = \{x \leftarrow 1\}$. Viewing these restrictions as conditional reductions we now have:

R4: If $x \neq 1$ and $y \neq 0$ and $z \neq 0$ then $x*(y+z) \rightarrow (x*y)+(x*z)$

R5: If $x \neq 1$ then $x*0 \rightarrow 0$

R6: If $x \neq 1$ then $x*(-y) \rightarrow -(x*y)$

The set {R1, R2, R3, R4, R5, R6} has been shown to be a complete set of reductions for commutative rings with unit elements relative to the ACI equational theory for + and *.

Example 3: As a final example let us examine a reduction which leads to a more complicated set of restrictions. Consider the following reduction which is an absorbtion law from the definition of a distributive lattice:

R7: $x + (x*y) \rightarrow x$

$I(\lambda) = \{x \leftarrow 0,\ x \leftarrow 1,\ y \leftarrow 0,\ y \leftarrow 1\}$. Note that $y \leftarrow 0$ must be included because, under identity substitution and coring, it is possible for y to appear in the scope of the + operator. $P(\lambda)$ with elements listed from smallest to largest is $\{\phi,\ \{x \leftarrow 0\},\ \{x \leftarrow 1\},\ \{y \leftarrow 0\},\ \{y \leftarrow 1\},\ \{x \leftarrow 0,\ y \leftarrow 0\},\ \{x \leftarrow 0,\ y \leftarrow 1\},\ \{x \leftarrow 1,\ y \leftarrow 0\},\ \{x \leftarrow 1,\ y \leftarrow 1\}\}$. Note that several members of $2^{I(\lambda)}$ were discarded because they were not valid substitutions. Of the remaining substitutions, only $\{x \leftarrow 0,\ y \leftarrow 1\}$ and $\{x \leftarrow 1,\ y \leftarrow 0\}$ are placed in $\Theta(\lambda \rightarrow \rho)$. This restriction differs from the previous examples in that it allows for either x or y to take on an identity, but prevents both x and y from taking on identities at the same time. Represented as a conditional reduction, R7 now becomes:

R7: If $\neg((x = 0 \text{ and } y = 1)$ or $(x = 1 \text{ and } y = 0))$ then $x + (x*y) \rightarrow x$,

or, equivalently,

R7: If $(x \neq 0 \text{ or } y \neq 1)$ and $(x \neq 1 \text{ or } y \neq 0)$ then $x + (x*y) \rightarrow x$.

2. Rewriting Strength.

Have we weakened the original rewriting relations by adding the conditions in the above examples? No, we have not. In Example 1 the most general form of a critical pair which could have been conflated by R3 before the conditions but cannot

be conflated by R3 after the conditions must be $< -(t+0),\ (-t)+(-0) >$ or $< -(0+t),\ (-0)+(-t) >$. It is easy to see that R1 can be used to conflate all such pairs since $(-t)+(-0) \stackrel{E}{=} (-t)+((-0)+0) \stackrel{R1}{\rightarrow} (-t)+0 \stackrel{E}{=} -(t+0)$. Thus, taken together, the rewriting power of {R1, R2, R3} has not been weakened by the introduction of the conditions needed for termination.

Likewise, in Example 2 we see that the most general form of pairs which could have been conflated by R4 were it not for the conditions must be either $< 1*(y+z),\ 1*y+1*z >$, $< x*(0+z),\ x*0+x*z >$, or $< x*(y+0),\ x*y+x*0 >$. The pair $< 1*(y+z),\ 1*y+1*z >$ conflates trivially since $1*(y+z) \stackrel{E}{=} 1*y+1*z$. The other two pairs are easily conflated via R5 since $x*0+x*z \stackrel{R5}{\rightarrow} 0+x*z \stackrel{E}{=} x*(0+z)$. As before we see that, taken together, the rewriting power of the entire reduction set has not been weakened by the conditions.

Finally, we see that in Example 3 the restriction on R7 only prevents its application to a pair of the general form $< 0+(0*1),\ 0 >$ or $< 1+(1*0),\ 1 >$, which conflate trivially since $0+(0*1) \stackrel{E}{=} 0$ and $1+(1*0) \stackrel{E}{=} 1$. Thus the restriction only prevents its application when its application was not needed in the first place. These examples indicate that the conditions needed for termination may not weaken the rewriting strength of a reduction at all, and that when they do another reduction in the same set may still provide the same functionality as that which was removed. In such cases termination is achieved while the set of reductions as a whole loses no rewriting strength. $\stackrel{R/E}{\rightarrow}$.

3. Implementing the Rewriting Relation.

Since $\stackrel{R/E}{\rightarrow}$ is a very general form of a rewriting relation between congruence classes generated by an equational theory, it is clear that the conditions which give $\stackrel{R/E}{\rightarrow}$ termination also give the termination of many less general rewriting relations. For

instance, any rewriting relation $\overset{R^E}{\rightarrow}$ such that $\overset{R}{\rightarrow} \subseteq \overset{R^E}{\rightarrow} \subseteq \overset{R/E}{\rightarrow}$ must terminate under these same conditions. This is important because the $\overset{R/E}{\rightarrow}$ rewriting relation is not conveniently implemented in a computer program, especially when E generates infinite congruence classes. We have found it useful to implement $\overset{R^E}{\rightarrow}$ for an ACI equational theory, E, as follows:

$$t_1 \underset{\lambda,\rho,\sigma}{\overset{R^E}{\rightarrow}} t_2 \quad \textit{iff} \quad t_1 \underset{\lambda,\rho,\sigma}{\overset{R,E}{\rightarrow}} t'_2 \overset{I,AC}{\dot{\rightarrow}} t'_2\downarrow^I \equiv t_2, \text{ and } \sigma \notin \iota_E\Theta(\lambda \rightarrow \rho).$$

This rewriting relation is in the range between $\overset{R}{\rightarrow}$ and $\overset{R/E}{\rightarrow}$ and is very easy to implement. The conditions which give termination are enforced as a simple modification to the ACI term matching routine. The term matching routine receives a term, a pattern, and the conditions. Whereas the normal ACI term matching would return the first substitution which matches the pattern to the term, the modified routine returns the first such match which does not violate the conditions. If no such match can be found, the term and pattern are considered not to match.

The rewriting relation we have described is actually a rewriting relation from a core element of one congruence class to a core element of another congruence class. When rewriting with in this manner, we begin with a core element, but we are allowed to leave the core during the ACI-matching step before we apply the reduction. We then push the result back down to the core.

V. ON ACI-COMPLETION

A. INTRODUCTION

The goal of this chapter is to generalize the theory of E-completion to enable the presentation of an ACI-completion procedure. In doing so we achieve three of the four benefits achieved by earlier generalizations: (1) the enhanced pattern matching process will be more akin to the human process since identity elements are the root of the problem in the mathematical theory and they are easily dealt with by the human mathematician, (2) increased step size for equality inferences results in smaller search trees, shorter proofs, and smaller reduction sets, (3) insight is provided into related problems, particularly the problem of E-completion for other equational theories which generate infinite congruence classes. We do not increase the problem domain for the universal word problem, however, because we have not found a complete set of reductions via ACI-completion for any algebraic structure not already handled via AC-completion.

The basic approach of this work will be to build around the generalized $\overset{R/E}{\rightarrow}$ rewriting relation presented in Chapter 4. This rewriting relation has been shown to terminate when E is an ACI equational theory. Because it is a generalization of the $\overset{R/E}{\rightarrow}$ rewriting relation used in previous E-completion procedures such as [*JK*86, *BP*87], we cannot assume their E-completion results, but must develop our ACI-completion procedure to match the new definition of $\overset{R/E}{\rightarrow}$.

B. CONDITIONAL REWRITING DEFINITIONS

Before proceeding to the ACI-completion theory, we first define conditional versions for each of the standard rewriting relations which we will use. As mentioned previously, each standard rewriting relation can be thought of as an instance of its conditional counterpart, with all of the conditions being empty. For terms t and s, we will say that $t \underset{\lambda,\rho,\sigma,m}{\overset{R}{\rightarrow}} s$ iff there is a reduction $\lambda \rightarrow \rho \in R$, $m \in dom(t)$, and a substitution σ such that

$$\sigma \notin \iota_E\Theta(\lambda \rightarrow \rho),$$

$$t/m = \lambda\sigma, \text{ and}$$

$$s = t[m \leftarrow \rho\sigma].$$

The restriction $\sigma \notin \iota_E\Theta(\lambda \rightarrow \rho)$ is the only difference between our relation and the usual definition of $\overset{R}{\rightarrow}$. The procedure for calculating $\Theta(\lambda \rightarrow \rho)$ is given in Chapter 4. We will say that $t \underset{\lambda,\rho,\sigma,m}{\overset{R,E}{\rightarrow}} s$ iff there is a reduction $\lambda \rightarrow \rho \in R$, $m \in dom(t)$, and a substitution σ such that

$$\sigma \notin \iota_E\Theta(\lambda \rightarrow \rho),$$

$$t/m \overset{E}{=} \lambda\sigma, \text{ and}$$

$$s = t[m \leftarrow \rho\sigma].$$

Equivalently, we may state this definition by

$$t \underset{\lambda,\rho,\sigma,m}{\overset{R,E}{\rightarrow}} s \text{ iff } t \underset{\leq}{\overset{E}{=}} t' \underset{\lambda,\rho,\sigma,m}{\overset{R}{\rightarrow}} s.$$

Finally, we define the rewriting relation $\overset{R/E}{\rightarrow}$ by

$$t \underset{\lambda,\rho,\sigma}{\overset{R/E}{\rightarrow}} s \text{ iff } t \overset{E}{=} t' \underset{\lambda,\rho,\sigma}{\overset{R}{\rightarrow}} s' \overset{E}{=} s.$$

Except for the conditions, these rewriting relations are just like their counterparts which are defined in Chapter 2. It is shown in Chapter 4 that all three of these

conditional rewriting relations satisfy the termination property when E is an ACI equational theory.

C. TESTING ACI-COMPLETENESS

The material presented in this section is a summary of material presented in [*PB*87] and much of it is taken verbatim from that source. The work done by Peterson et al. is closely related to this work and lays the essential groundwork for the remainder of this paper. We summarize only necessary results here, omitting the proofs.

1. E-Church-Rosser Property.

The theoretical basis for this section is a general E-Church-Rosser theorem similar to that of [*JK*86], using similar notation which we now review. Let S be a set. Let $\overset{E}{=}_1$ be a symmetric relation on S and let $\overset{E}{=}$ be its reflexive, transitive, closure. Let R (or $\overset{R}{\rightarrow}$) be a relation on S and R/E be the relation $\overset{E}{=} \circ \overset{R}{\rightarrow} \circ \overset{E}{=}$, which must be well-founded. Let $t{\downarrow}^R$ be a normal form obtained from t using the well-founded relation R. Let R^{-1} (or $\overset{R}{\leftarrow}$) be the set $\{(p, q) \mid (q, p) \in R\}$. Let $\overset{R \cup E}{=}_1$ be the relation $\overset{E}{=}_1 \cup \overset{R}{\rightarrow} \cup \overset{R}{\leftarrow}$, and let $\overset{R \cup E}{=}$ be the reflexive, transitive, closure of $\overset{R \cup E}{=}_1$. Finally, let R^E (or $\overset{R^E}{\rightarrow}$) be any relation which satisfies the inequality $R \subseteq R^E \subseteq R/E$. We now make the following definitions:

Definition 1: R^E is *E-Complete* means

$$s \overset{R \cup E}{=} t \text{ iff } s{\downarrow}^{R^E} \overset{E}{=} t{\downarrow}^{R^E}.$$

We will use the notation $< s \overset{R \cup E}{=} t >$ throughout this paper to represent critical pairs, rather than the traditional notation $< s, t >$. This will serve as a reminder that the

process of forming a critical pair generates two terms whose normal forms must be brought together in order to achieve E-completeness.

Definition 2: R^E is *locally coherent modulo* E if whenever $t \overset{E}{=}_1 s$ and $t \overset{R^E}{\rightarrow} t_1$, it follows that there is a term s_1 such that $s \overset{R^E}{\rightarrow} s_1$ and t_1 and s_1 have a common $\overset{R/E}{\rightarrow}$ successor.

Definition 3: R^E is *locally confluent modulo* E if whenever $t \overset{R^E}{\rightarrow} t_1$ and $t \overset{R^E}{\rightarrow} t_2$, it follows that t_1 and t_2 have a common $\overset{R/E}{\rightarrow}$ successor.

E-Church-Rosser Theorem: [*PB*87]

The following two statements are equivalent:

(1) R^E is E-complete.

(2) R^E is locally coherent and locally confluent modulo E.

2. Local Coherence Property.

We now state a characterization of local coherence for our conditional rewriting relation.

Local Coherence Theorem: [*PB*87]

The following two statements are equivalent:

(1) R, E is locally coherent modulo E.

(2) Whenever $l = r \in E$ (or $r = l \in E$), $\lambda \rightarrow \rho \in R$, $m \in sdom(l)$ but $m \neq \varepsilon$, and $\sigma \in E\text{-}Unify(l/m, \lambda) \ni \sigma \notin i_E\Theta(\lambda \rightarrow \rho)$, it follows that $\exists u \ni r\sigma \overset{R,E}{\rightarrow} u$ and $l\sigma[m \leftarrow \rho\sigma]$ and u have a common $\overset{R/E}{\rightarrow}$ successor.

We will say that $\lambda \rightarrow \rho$ *coheres* with $l = r$ when (2) above holds for all appropriate values of m and σ.

Based on the Local Coherence Theorem we can implement the following test for local coherence: First find all $l = r$, $\lambda \rightarrow \rho$, m, and σ which satisfy the conditions of (2) of the theorem. Then test each $r\sigma$ for $\overset{R,E}{\rightarrow}$ reducibility. If some $r\sigma$ is not reducible

then coherence fails. If $r\sigma$ is reducible by $\overset{R,E}{\rightarrow}$ using some $\lambda' \rightarrow \rho'$, σ', and m', then the *coherence critical pair*

$$< l\sigma[m \leftarrow \rho\sigma] \overset{R \cup E}{=} r\sigma[m' \leftarrow \rho'\sigma'] >$$

can be reduced to normal form via $\overset{R/E}{\rightarrow}$. If both sides are identical in some normal form, then coherence succeeds, otherwise coherence fails.

When the set E is an ACI equational theory, which is our interest in this research, the test for local coherence can be further simplified by the following observations:

(1) All reductions cohere with the identity laws. This is true because for an arbitrary identity law, $x + 0 = 0$, the only m satisfying (2) of the theorem is the one such that $l/m = 0$, thus $\sigma \in E\text{-}Unify(\lambda, 0)$. This means that $\lambda\sigma \overset{E}{=} 0$, however, it is shown in Chapter 4 that $\lambda\sigma \overset{E}{=} 0 \Rightarrow \sigma \in \iota_E\Theta(\lambda \rightarrow \rho)$ by the definition of $\Theta(\lambda \rightarrow \rho)$ and necessary properties of the weighting function.

(2) All reductions cohere with the commutative laws. This is true because when $l = r$ is a commutative law there is no m satisfying $m \in sdom(l)$ and $m \neq \varepsilon$.

(3) If $\lambda \rightarrow \rho \overset{ACI}{=} w + \lambda' \rightarrow w + \rho'$ where $w \notin Vars(\lambda' \rightarrow \rho')$ then $\lambda \rightarrow \rho$ coheres with the associative law for $+$. This is proved in [*PB*87] under the added assumption that $w \notin Vars(\Theta(\lambda \rightarrow \rho))$. From the calculation procedure given for $\Theta(\lambda \rightarrow \rho)$ in Chapter 4, however, we can show that the conditions assumed here imply that $w \notin Vars(\Theta(\lambda \rightarrow \rho))$. The essence of this observation is that reductions which have already been extended for a given ACI operator automatically cohere with the associative law for that operator.

Since coherence with the identity and commutative laws is automatic and extensions give coherence with the associative laws, the procedure for assuring coherence for an ACI equational theory simplifies to the following: for each

reduction $\lambda \to \rho \in R$ together with the associative law for each ACI operator + in λ perform the coherence test described above. If the test fails we can then replace $\lambda \to \rho$ with its extension, $w + \lambda \to w + \rho$ such that $w \notin Vars(\lambda \to \rho)$, and coherence will be assured.

Because the Local Coherence Theorem only requires a common $\overset{R/E}{\to}$ successor, we are allowed to flatten and/or core the terms after the coherence critical pair has been formed, during the process of finding normal forms. This is because the E-equality steps involved in the flattening and coring processes together with the $\overset{R/E}{\to}$ reductions still form a rewriting relation which is consistent with the definition of $\overset{R/E}{\to}$. It is very important, however, that this flattening and/or coring not take place until after the coherence critical pair has been formed because the theorem calls for $r\sigma$ to be $\overset{R,E}{\to}$ reducible and flattening and/or coring $r\sigma$ too early results in a test for $\overset{R/E}{\to}$ reducibility instead. In actuality, we usually use the rewriting relation $\overset{R,E}{\to}$ together with flattening and coring to produce normal forms which are tested for E-equality, because this is much easier to implement than the more general rewriting relation $\overset{R/E}{\to}$. Clearly this is still a valid process as long as we use a rewriting relation which is contained in $\overset{R/E}{\to}$.

3. Local Confluence Property.

We now state a characterization of local confluence based on our conditional rewriting relation.

Local Confluence Theorem: [*PB*87]

If R,E is locally coherent modulo E, then the following two statements are equivalent.

(1) R^E is locally confluent modulo E.

(2) Whenever $\lambda_1 \to \rho_1 \in R$, $\lambda_2 \to \rho_2 \in R$, $m \in sdom(\lambda_1)$, and $\sigma \in E\text{-}Unify(\lambda_1/m, \lambda_2)$ $\ni$ $\sigma \notin \iota_E\Theta(\lambda_1 \to \rho_1)$ and $\sigma \notin \iota_E\Theta(\lambda_2 \to \rho_2)$, it follows that $\lambda_1\sigma[m \leftarrow \rho_2\sigma]$ and $\rho_1\sigma$ have a common $\overset{R/E}{\to}$ successor.

Based on the Local Confluence Theorem we can implement the following test for confluence: First ensure that local coherence is satisfied. This is done using the procedure described with the Local Coherence Theorem. In the case of an ACI equational theory we know that we will be able to satisfy coherence by extending reductions where needed, as pointed out above. The next step is to generate all of the *confluence critical pairs*

$$< \lambda_1\sigma[m \leftarrow \rho_2\sigma] \overset{R \cup E}{=} \rho_1\sigma >$$

for which the hypotheses of (2) above are satisfied. Both sides of the critical pair are then reduced to normal forms via $\overset{R/E}{\rightarrow}$ and compared. If the normal forms are identical, the pair *conflates*, and the test for confluence succeeds, otherwise it fails. Again we remark that it is sufficient to compute normal forms via $\overset{R,E}{\rightarrow}$ together with flattening and/or coring, with a final check for E-equality.

4. An Algorithm to Test ACI-Completeness.

Figure 13 presents an algorithm which applies the Local Coherence Theorem and the Local Confluence Theorem together with the the coherence results relating specifically to ACI equational theories to test a set of reductions for ACI-completeness. This algorithm has been implemented in a computer program and has verified several sets of reductions to be ACI-complete. The ACI-completion procedure presented in the final section of this paper will be a generalization of this algorithm.

The function *E-Unify* may be any finite E-unification algorithm which returns a complete set of E-unifiers for the given terms, such as the one presented in Chapter 2. The function *ACI-Operators* needs to return all of the ACI operators which appear in

the given term. The functions *Vars* and *Sdom* are both defined in Chapter 2. The restricted substitution function Θ is described in Chapter 4.

Procedure **Test-ACI-Completeness** (R)
  For each $\lambda \rightarrow \rho \in R$ do
    For each $+ \in ACI\text{-}Operators(\lambda)$ do
      $a :=\ (x + (y + z) = (x + y) + z)$
      If $\exists (s = t) \in Coherence\text{-}Critical\text{-}Pairs(a, \lambda \rightarrow \rho) \ni s\downarrow^{R/E} \overset{E}{\neq} t\downarrow^{R/E}$
      Then $R :=\ R - \{\lambda \rightarrow \rho\}$
           $R :=\ R \cup \{w + \lambda \rightarrow w + \rho\},\ w \notin \text{Vars}(\lambda \rightarrow \rho)$
    End For
  End For
  If $\exists (s = t) \in \cup_{(p,q) \in R \times R} Confluence\text{-}Critical\text{-}Pairs(p,q) \ni s\downarrow^{R/E} \overset{E}{\neq} t\downarrow^{R/E}$
  Then Return *Failure*
  Else Return *Success*
End Procedure Test-ACI-Completeness

Procedure **Coherence-Critical-Pairs** ($l = r, \lambda \rightarrow \rho$)
  Return $\{(r\sigma = \rho\sigma) \mid \sigma \in E\text{-}Unify(l, \lambda),\ \sigma \notin \iota_E\theta(\lambda \rightarrow \rho)\}$
    $\cup \{(r\sigma = (l[m \leftarrow \rho])\sigma) \mid \sigma \in E\text{-}Unify(l/m, \lambda),\ \sigma \notin \iota_E\theta(\lambda \rightarrow \rho),$
      $m \in Sdom(l),$ and $m \neq \varepsilon\}$
    $\cup \{((\lambda[m \leftarrow r])\sigma = \rho\sigma) \mid \sigma \in E\text{-}Unify(l, \lambda/m),\ \sigma \notin \iota_E\theta(\lambda \rightarrow \rho),$
      $m \in Sdom(\lambda),$ and $m \neq \varepsilon\}$
End Procedure Coherence-Critical-Pairs

Procedure **Confluence-Critical-Pairs** ($\lambda_1 \rightarrow \rho_1, \lambda_2 \rightarrow \rho_2$)
  Return $\{(\rho_1\sigma = \rho_2\sigma) \mid \sigma \in E\text{-}Unify(\lambda_1, \lambda_2),\ \sigma \notin \iota_E\theta(\lambda_1 \rightarrow \rho_1),$
      and $\sigma \notin \iota_E\theta(\lambda_2 \rightarrow \rho_2)\}$
    $\cup \{(\rho_1\sigma = (\lambda_1[m \leftarrow \rho_2])\sigma) \mid \sigma \in E\text{-}Unify(\lambda_1/m, \lambda_2),\ \sigma \notin \iota_E\theta(\lambda_1 \rightarrow \rho_1),$
      $\sigma \notin \iota_E\theta(\lambda_2 \rightarrow \rho_2),\ m \in Sdom(\lambda_1),$ and $m \neq \varepsilon\}$
    $\cup \{((\lambda_2[m \leftarrow \rho_1])\sigma = \rho_2\sigma) \mid \sigma \in E\text{-}Unify(\lambda_1, \lambda_2/m),\ \sigma \notin \iota_E\theta(\lambda_1 \rightarrow \rho_1),$
      $\sigma \notin \iota_E\theta(\lambda_2 \rightarrow \rho_2),\ m \in Sdom(\lambda_2),$ and $m \neq \varepsilon\}$
End Procedure Confluence-Critical-Pairs

Figure 13. An algorithm to test ACI-Completeness

D. ACI-COMPLETION CONSIDERATIONS

As we develop an ACI-completion procedure based on the test for ACI-completeness presented in the previous section we will first address three areas where the ACI-completion procedure differs from previous E-completion procedures. These differences arise because we are focusing specifically on an ACI equational theory and because we are working with the conditions needed for termination of the $\overset{R/E}{\rightarrow}$ rewriting relation.

1. Identity Substitution Inference.

The first concept peculiar to ACI-completion which we would like to address is what we have called *identity substitution inference*. By the Compatibility Lemma given in Chapter 2 we know that if $l \overset{R \cup E}{=} r$ is a valid equation, then for any substitution, σ, $l\sigma \overset{R \cup E}{=} r\sigma$ is also a valid equation. We are only interested in the case where σ is a substitution which replaces one or more variables in l and/or r with an identity. Having made the identity substitutions we then core each side, still preserving $(R \cup E)$-equality, and obtain a new equation, $(l\sigma)\downarrow^{I} \overset{R \cup E}{=} (r\sigma)\downarrow^{I}$.

An application of identity substitution inference is found in the processing of confluence critical pairs. When a confluence critical pair is found which will not conflate via the existing R and no orientation can be found to use the pair as a reduction, identity substitution inference may be helpful in resisting failure. Similar to other failure resistance schemes which delay the processing of a troublesome pair, hoping to be able to handle it after other pairs have been processed and more reductions have been added to R [*FG*84], we put the problem pair aside for later processing. Rather than simply going on to other pairs before coming back to the problem pair, however, we first try the rule of identity substitution inference. This is

an application of the idea that learning something is better than learning nothing at all.

Suppose we have a critical pair, $< l \overset{R \cup E}{=} r >$ which will neither conflate nor orient to form a reduction. Suppose further that we can find an identity substitution, σ, such that the new equation, $(l\sigma)\downarrow^{I} \overset{R \cup E}{=} (r\sigma)\downarrow^{I}$, is orientable as a reduction. Then we can add the new reduction to R, possibly increasing our chances of conflating the problem pair when we come back to it. In fact, it may be the case that the reduction formed by the identity substitution inference is immediately able to conflate the critical pair from which it was generated. The hope that identity substitution and coring will produce an orientable equation from an unorientable one is based on the possibility that some variable which is replaced by an identity may occur in a different context on one side of the equation than it does on the other, causing one side to "collapse" more than the other.

The following example illustrates how identity substitution inference may be used to resist failure during the ACI-completion process. Let E be the ACI equational theory for $+$ and let the initial set R of reductions contain only the single reduction

$$\text{R1: } x + y + (-y) \rightarrow x.$$

Noting that this reduction passes the test for local coherence, we move to the test for local confluence. Forming confluence critical pairs of R1 with itself we obtain the following pairs:

$$\text{P1: } < x \overset{R \cup E}{=} x >$$

$$\text{P2: } < x + y \overset{R \cup E}{=} x + y >$$

$$\text{P3: } < -(x + y) + y + z \overset{R \cup E}{=} -(x + w) + w + z >$$

$$\text{P4: } < x + y \overset{R \cup E}{=} -(-(z + x) + z + w) + w + y >$$

P1 and P2 obviously conflate without applying any reductions. The pair P3, however, does not conflate since the sides are not E-equal and the only reduction, R1, cannot

be applied. Furthermore, it is clear from the form of each side that any weighting function which weights all variables equally will assign the same weight to both sides of the pair, preventing us from orienting the sides to form a reduction. At this point we put P3 aside for later processing and see if we can gain anything from the rule of identity substitution inference. Applying the substitution $\sigma = \{w \leftarrow 0\}$ to both sides of the pair and coring the result we get the new pair

$$\text{P5:} \quad < -(x+y)+y+z \overset{R\cup E}{=} (-x)+z > .$$

This pair is now orientable as a reduction giving us

$$\text{R2:} \quad -(x+y)+y+z \rightarrow (-x)+z$$

which is added to R. Recalling that we have set the pair P3 aside for later processing, we note that P3 will now conflate since both sides rewrite to $(-x)+z$ using R2. Thus we have gained enough information from the identity substitution inference to completely handle the P3 pair.

Another application of identity substitution inference is the removal of unnecessary variables from reductions. Suppose that we have generated the critical pair

$$\text{P6:} \quad < x+-(-y) \overset{R\cup E}{=} x+y > ,$$

the pair will neither conflate nor reduce further by the present R, and the weighting function assigns a greater weight to the left hand side of the pair. We could, of course, form the reduction

$$\text{R3:} \quad x+-(-y) \rightarrow x+y.$$

If we note, however, that the reduction

$$\text{R4:} \quad -(-y) \rightarrow y$$

will also conflate the pair P6, then we can simply apply the rule of identity substitution inference to the critical pair using the substitution $\sigma = \{x \leftarrow 0\}$ and form the reduction R4 instead. Furthermore, it is easy to verify that R4 will pass all the

tests for local coherence for any ACI equational theory. In other words, the completion process was about to generate a reduction with an unneeded variable.

The procedure just described can easily be generalized. Before forming a reduction $\lambda \rightarrow \rho$ from a non-conflating critical pair p, find the largest identity substitution σ such that $(\lambda\sigma)\downarrow^I \rightarrow (\rho\sigma)\downarrow^I$ will conflate p. Add the smaller reduction $(\lambda\sigma)\downarrow^I \rightarrow (\rho\sigma)\downarrow^I$ to the reduction set instead of $\lambda \rightarrow \rho$. The significance of removing these extraneous variables before the reduction is formed is that this helps to keep down the number of variables involved in the ACI-unification and ACI-matching processes as well as the number of overlappings involved in computing critical pairs. We have observed considerable savings of both time and space during the ACI-completion process using this method.

2. Satisfying Coherence.

From the E-Church-Rosser Theorem we have seen that both local confluence and local coherence are needed to have an E-complete reduction set. Local confluence has always been the central issue of completion and E-completion procedures up to this point. If confluence pairs do not conflate, new reductions are formed and added to the reduction set. No reduction is ever removed from the reduction set unless some combination of the other reductions provides duplicate functionality. This guarantees that the reduction set can only increase in rewriting strength during the completion process, and that the final reduction set will have the capability of conflating all critical pairs which were generated at any point during the completion process. We propose to address the local coherence property by processing coherence critical pairs in precisely the same manner in which confluence critical pairs are normally processed.

When a reduction is added to the reduction set we simply compute all coherence critical pairs between that reduction and the equational theory, placing these new critical pairs on the list with any outstanding confluence and/or coherence critical pairs. If the program halts, we will know that all confluence pairs of the final reduction set and all coherence pairs of the final reduction set have already been conflated. Thus both properties are satisfied and the reduction set is E-complete.

This completely avoids using the concept of extended reductions. In turn, this also eliminates the need for protection schemes which must be used to ensure that the form of an extended reduction is not altered in such a way as to lose the coherence property and to ensure that a reduction which is needed for the coherence of another reduction is not deleted from the reduction set. We will address the issue of extensions and protection schemes more in the next chapter. It is interesting to note that most of the reductions which are formed as the result of processing coherence critical pairs in the manner suggested above are exactly the same as the extended reductions which come from the theory of extensions. The exceptions to this rule are precisely those cases where the extension would not function properly without the protection scheme. Of course, for those reductions which already satisfy the coherence property with respect to the equational theory, all coherence pairs will ultimately conflate without causing the addition of any new reductions.

3. Critical Pairs and Conditional Reductions.

There are two ways in which conditional reductions affect the processing of critical pairs. The first effect is found in the computation of the pairs themselves. As indicated by the Local Coherence and Local Confluence Theorems, critical pairs are affected by the conditions on both of the involved terms. Because the unifiers generated by overlapping the two terms must not violate the conditions of either term, many potential critical pairs are avoided.

The second effect of conditions on the processing of critical pairs is seen in the processing of a confluence critical pair which does not conflate. In all previous completion and E-completion procedures, critical pairs which do not conflate are oriented, if possible, to form a new reduction which is added to R. Because of the conditions which may be required to maintain termination of the rewriting relation, however, we cannot simply orient the pair and add a reduction to R. Once we have determined an orientation of $\lambda \rightarrow \rho$, we must compute the conditions which ensure termination, according to the procedure given in Chapter 4. These conditions are calculated in the form of a set of restricted substitutions, $\Theta(\lambda \rightarrow \rho)$, such that no rewriting is allowed using $\lambda \rightarrow \rho$ and a term matching substitution which is an E-instance of a substitution in the restricted set. If $\Theta(\lambda \rightarrow \rho)$ is empty then we can simply add the new reduction to R and procede as most other E-completion procedures. If $\Theta(\lambda \rightarrow \rho)$ is not empty, however, then we will handle the critical pair according to a method which we will call *splitting a critical pair*.

The concept of splitting a critical pair is based on the concept of proof by exhaustive cases. If we need to show that $\forall x\, P(x)$ is true, we know that this can be done by splitting the proof into exhaustive cases and then showing that the desired result holds for each case. For instance, if we are able to show that $P(0)$ is true and that $\forall x\, x \neq 0 \Rightarrow P(x)$ is true, then we have shown that $\forall x\, P(x)$ is true. A simple example illustrates how we can apply this to a critical pair. Suppose we have just generated the critical pair

$$\text{P7:} \quad < x*0 \overset{R \cup E}{=} 0 >$$

and it will not conflate via the current R. Orienting the sides by weight gives the reduction $x*0 \rightarrow 0$ and calculating the restricted substitutions gives $\Theta(\lambda \rightarrow \rho) = \{x \leftarrow 1\}$. In conditional form this gives a potential reduction of

$$\text{R5:} \quad \text{If } x \neq 1 \text{ then } x*0 \rightarrow 0.$$

In order to satisfy the test for local confluence which is needed to have a complete set of reductions, however, we must show that the pair P7 conflates for all values of x. Clearly the reduction R5 will cause the pair P7 to conflate for all values of x except the case when $x = 1$. We thus split the critical pair into the reduction R5 and the new critical pair

$$\text{P8:} \quad < 1*0 \overset{R \cup E}{=} 0 >$$

which is formed by applying the substitution $\{x \leftarrow 1\}$ to the P7 pair. The new pair, P8, represents the only instance of P7 not conflated by R5. This new pair must also be conflated before the reduction set is deemed to be complete. The hope is that the new pair will either conflate trivially (without the application of any reductions), conflate via other reductions in R, or orient to form yet another reduction. In this example the P8 pair conflates trivially because $1*0 \overset{E}{=} 0$, showing us that R5, even with its restrictions, is sufficient to conflate the P7 pair.

The process of splitting a critical pair becomes slightly more complicated as the restricted substitution set grows in size. For example the critical pair

$$\text{P9:} \quad < -(x+y) \overset{R \cup E}{=} (-x) + (-y) >$$

gives $\Theta(\lambda \rightarrow \rho) = \{\{x \leftarrow 0\},\{y \leftarrow 0\}\}$. As a conditional reduction we have

$$\text{R6: If } x \neq 0 \text{ and } y \neq 0 \text{ then } -(x+y) \rightarrow (-x) + (-y).$$

In order to show that the P9 pair conflates in all cases we must show that it conflates for all values of x and y. Clearly this can be done by showing that P9 conflates in the following three cases which exhaust the domain of x and y.:

Case 1: $x \neq 0$ and $y \neq 0$

Case 2: $x = 0$

Case 3: $y = 0$

The P9 pair conflates for Case 1 via the conditional reduction, R6. P9 conflates for Case 2 and Case 3 if the critical pairs

$$\text{P10:} \quad < -(0+y) \overset{R \cup E}{=} (-0) + (-y) >, \text{ and}$$

$$\text{P11:}\quad <-(x+0) \overset{R\cup E}{=} (-x)+(-0)>,$$

respectively, conflate. P10 and P11 were formed by separately applying each of the substitutions in $\Theta(\lambda \rightarrow \rho)$ to P9. As before, the new critical pairs which are split off of the original critical pair include all instances of the original which are not conflatable via the new conditional reduction.

Generalizing the previous examples gives us the following procedure for splitting a critical pair, p, which does not conflate: First we orient the sides of the pair and compute $\Theta(\lambda \rightarrow \rho) = \{\theta_1, \theta_2, ..., \theta_n\}$. We then add the conditional reduction $[\Theta(\lambda \rightarrow \rho), \lambda \rightarrow \rho]$ to R and add the new critical pairs $p\theta_1$, $p\theta_2$, ..., $p\theta_n$ to the list of critical pairs which must be processed independently. The pair p must be conflated for all values of its variables, or equivalently, $p\sigma$ must conflate for all substitutions σ. Clearly the new reduction will conflate $p\sigma$ for all substitutions σ except those which are E-instances of some θ_i. The pair $p\theta_i$, however, represents the most general form of an instance of $p\sigma$ not handled by the new reduction. If all pairs $p\theta_i$ conflate then $p\sigma$ conflates for all substitutions σ. Note that when $\Theta(\lambda \rightarrow \rho)$ is empty no new critical pairs are formed, only a reduction. This is exactly the manner in which a non-conflating critical pair is handled in previous completion and E-completion procedures.

E. AN ACI-COMPLETION PROCEDURE

Figure 14 presents a procedure which attempts to complete a set of reductions modulo an ACI equational theory. This procedure is a generalization of the algorithm for testing ACI-completeness which was presented earlier. It follows the pattern of the AC-completion procedure presented in Chapter 3 with the following enhancements:

(1) failure resistance is built in according to the method of [*FG*84]

(2) identity substitution inference is added to the failure resistance mechanism,

(3) the rewriting relation uses conditional reductions to maintain termination in the presence of infinite ACI congruence classes,

(4) coherence critical pairs are processed just as confluence critical pairs, without the use of extensions, and

(5) the process of splitting critical pairs is used to ensure confluence via conditional reductions.

In the completion procedure given in Figure 14 the following conventions are used: R is a set of conditional reductions; RR is that portion of $R \times R$ which has not been processed; CP is a set of critical pair equations; and D is a set of delayed critical pairs, pairs which have been put aside in order to avoid failure until all other pairs have been processed.

The top level procedure *ACI-Completion* implements the failure resistance layer. This routine is normally invoked with an empty R and the defining equations (other than the ACI equations) in CP. When the defining equations, E_1, form a superset of another set of equations, E_2, for which we have already found an ACI-complete set of reductions, we will invoke this procedure with the complete set of reductions for E_2 in R and $E_1 - E_2$ in CP.

The procedure *Try-To-Complete-R* is the normal E-completion layer. This procedure exhausts all of $R \times R$ before succeeding or giving up. The procedure *Try-To-Conflate-Pairs* exhausts the set of critical pairs before returning to the previous level. The procedure *Try-To-Form-Reduction* implements the concepts of resisting failure via identity substitution inference, and failure resistance via the delaying of a critical pair. This procedure returns an oriented reduction if one can be derived from the pair, and possibly a critical pair whose processing should be delayed as long as possible, indicating the original pair did not orient to form a reduction.

```
Procedure ACI-Completion (R, CP)
   Repeat
      R', CP := Try-To-Complete-R(R,CP)
   Until R' = R
   If CP = φ
   Then Return Success, R
   Else Return Failure
End Procedure ACI-Completion


Procedure Try-To-Complete-R (R, CP)
   RR, D := φ, φ
   While (RR ∪ CP) ≠ φ do
      R, RR, D' := Try-To-Conflate-Pairs(R,RR,CP)
      D := D ∪ D'
      If RR ≠ φ
      Then (r1, r2) := "smallest" member of RR
           RR := RR − {(r1, r2)}
           CP := Confluence-Critical-Pairs(r1, r2)
   End While
   Return R, D
End Procedure Try-To-Complete-R


Procedure Try-To-Conflate-Pairs (R, RR, CP)
   D := φ
   While CP ≠ φ do
      (s = t) := "smallest" member of CP
      CP := CP − {(s = t)}
      If s↓R/E ≠ t↓R/E
      Then r, d := Try-To-Form-Reduction(s↓R/E, t↓R/E)
           D := D ∪ {d}
           If r ≠ φ
           Then R,RR,CP := Add-Reduction(r,R,RR,CP)
                R,RR,CP := Simplify-R(R,RR,CP)
   End While
   Return R, RR, D
End Procedure Try-To-Conflate-Pairs
```

Figure 14A. An ACI-completion procedure - Part 1

The procedure *Add-Reduction* adds the condition to the oriented pair, adds critical pairs representing instances of the original which are not handled by the conditional reduction, and adds all of the coherence critical pairs formed from the

reduction and any relevant associative law. The procedures *Add-Reduction* and *Simplify-R* are used to keep the set R completely inter-reduced during the completion process. These procedures also keep RR current with R.

The procedures *Coherence-Critical-Pairs* and *Confluence-Critical-Pairs* are exactly the same as defined in Figure 13, page 78. The functions *ACI-Operators*, *Vars*, and *Sdom* are also the same as those presented in Figure 13. The function *AC-Operators* returns AC operators, just as *ACI-Operators* returns ACI operators. The weighting function W and the restricted substitution function Θ are both described in Chapter 4.

As with other completion procedures, this procedure may halt after finding an ACI-complete set of reductions; halt with failure after finding a non-conflating critical pair which cannot be oriented, even after all other potential critical pairs have been processed; or continue indefinitely adding new reductions to R. This procedure has been implemented in a computer program which has found ACI-complete reduction sets for a number of algebraic structures. Several of these will be presented in the next chapter.

The ACI-completion procedure presented in Figure 14 is not only able to perform E-completion relative to ACI equational theories, but also for empty, C, and AC equational theories as well as any combination of these. All that is necessary is that the procedure *E-Unify* return a finite and complete set of E-unifiers and that we are able to compute terminal forms via E-matching with respect to the desired equational theories. We point out that for empty and C equational theories there will be no coherence critical pairs, and for AC equational theories the only coherence critical pairs will be those from the associative law, just as those for ACI equational theories. Note that this is already handled in the *Add-Reduction* procedure. The conditions on the reductions will always be empty when a reduction has no ACI

```
Procedure Try-To-Form-Reduction (s, t)
   Case W(s) > W(t):
            Return s → t, φ
        W(t) > W(s):
            Return t → s, φ
        W(t) = W(s):
            If ∃(s' → t') ∈ {(sτ↓^I → tτ↓^I) | W(sτ↓^I) > W(tτ↓^I)
                                and ∀(x ← i) ∈ τ, i is an identity}
            Then Return s' → t', (s = t)
            Else Return φ, (s = t)
End Procedure Try-To-Form-Reduction


Procedure Add-Reduction (λ → ρ, R, RR, CP)
   r := (θ(λ → ρ), λ → ρ)
   R := R ∪ {r}
   RR := RR ∪ {(r, r') | r' ∈ R}
   CP := CP ∪ {(λ = ρ)τ | τ ∈ θ(λ → ρ)}
   A := {(x + (y + z) = (x + y) + z) | + ∈ ACI-Operators(λ) ∪ AC-Operators(λ)}
   CP := CP ∪ ∪_{a∈A} Coherence-Critical-Pairs(a,r)
   Return R, RR, CP
End Procedure Add-Reduction


Procedure Simplify-R (R,RR,CP)
   For each r ∈ R do
      If r↓^{(R-{r})/E} ≠^E r
      Then RR := RR − {(r, r') | r' ∈ R}
           R := R − {r}
           R, RR, CP := Add-Reduction(r↓^{(R-{r})/E}, R, RR, CP)
           R,RR,CP := Simplify-R(R,RR,CP)
   End For
   Return R, RR, CP
End Procedure Simplify-R
```

Figure 14B. An ACI-completion procedure - Part 2

operators, but this is permissible since they are often empty even when reductions do have ACI operators. Thus, with no changes, this one procedure not only provides E-completion for a new class of equational theories, but also for these important classes of equational theories which have been addressed by earlier researchers. We will demonstrate this generality in the upcoming chapter.

VI. RESULTS OF AN IMPLEMENTATION

The ACI-completion procedure given in the previous chapter has been implemented in a computer program. This program is written in Common LISP as described by Steele [*St*84] and has been run successfully under a number of different hardware and software configurations including a DEC Microvax II, a XEROX 1109, a Symbolics, and an IBM PC-RT. Those interested may obtain a copy of the program in machine readable form by contacting the author through the Computer Science Department, at the University of Missouri-Rolla. The purpose of this chapter is twofold. First we discuss important aspects of the program implementation which are not spelled out by the ACI-completion procedure given previously. Secondly, we present some of the complete sets of reductions which have been generated by this program.

A. IMPLEMENTATION NOTES

The following details of program implementation seem worthy of separate discussion: (1) the data structures used, (2) the implementation of the E-matching algorithm, (3) a method for dealing with term symmetry, (4) the use of extended reductions, and (5) the user interface. Each item has been selected because of its significance to the overall understanding and/or usage of the program. We now discuss each in turn.

1. Data Structures.

Constants, variables, and operators are represented by LISP atoms. Variables are distinguished from constants and operators by the value of the property *VARIABLE* on the LISP property list for the atom. For variables this property will have a value of *T*, for all other atoms the value will be *NIL*. Simple terms are

constants and variables. A complex term is represented as a list in prefix form, i.e. the first element of the list is always the operator. For example, the term $x + (-x)$ would be represented by the list $(+\ x\ (-\ x))$. AC and ACI operators result in *flattened* terms, i.e. $(+\ (+\ a\ b)\ c)$ is represented as $(+\ a\ b\ c)$, where + becomes an operator of varying degree.

The equational theory associated with an operator will be indicated by assigning a value of *T* to one of the properties *C*, *AC*, or *ACI* on the property list for the atom. If the *C* property is set to *T* the associated equational theory is understood to be the commutative theory for that operator. Likewise, *T* for the property *AC* indicates an associative/commutative equational theory and *T* for the property *ACI* indicates an associative/commutative/identity equational theory. No more than one of these properties should be set to *T*. All properties not set to *T* should be set to *NIL*. If all three of these properties are set to *NIL* the associated equational theory is empty. For an ACI operator, one additional property, *IDENTITY*, holds the value of the identity which is associated with the operator. For example, if + is an ACI operator with identity 0, the LISP function (GET '+ 'ACI) should return *T* and the LISP function (GET '+ 'IDENTITY) should return 0. Using the LISP property list to encode the associated equational theory makes this information readily available to any routine which needs to make use of it.

Substitutions are represented as lists of *variable* ← *term* pairs such that the LISP CAR of the pair gives the variable and LISP CDR of the pair gives the term. Because of this a substitution of a simple term for a variable results in a LISP dotted pair, while the substitution of a complex term for a variable results in a normal list whose second element is the primary operator for the term. For example, the substitution $\{x \leftarrow a,\ y \leftarrow (b + z)\}$ would be represented as the list $((x.a)\ (y\ +\ b\ z))$. The LISP atom *NIL* represents the empty substitution.

Because the restricted substitution set for a reduction never changes, we calculate $\Theta(\lambda \rightarrow \rho)$ according to the procedure given in Chapter 4 and store it with the reduction when the reduction is formed. This makes the restrictions available without calculation at any point where they are needed. A conditional reduction is thus represented as a list of three items: the restricted substitution set, the left hand side, and the right hand side. The unconditional reduction $(-(-x)) \rightarrow x$ would be stored as $(NIL\ (-\ (-\ x))\ x)$ where NIL indicates there are no restricted substitutions. The conditional reduction

$$\text{If } x \neq 0 \text{ and } y \neq 0 \text{ then } -(x+y) \rightarrow (-x)+(-y)$$

is stored as the list

$$(\ (((x.0))\ ((y.0)))\ (-\ (+\ x\ y))\ (+\ (-\ x)\ (-\ y))\ ).$$

Critical pairs and the defining equations for the algebraic structure which are not part of the equational theory are simply stored as lists of two items: the left hand side and the right hand side. Because these are equations and not reductions, however, there is no real significance as to which term is on which side. For example, the critical pair $x+(-(y+x)) \overset{R\cup E}{=} (-y)$ could be stored as either the list $(\ (+\ x\ (-\ (+\ y\ x)))\ (-y)\ )$ or its reverse.

All sets are represented as lists where the order is not significant. Thus the set $\{a,\ b,\ c\}$ may be any permutation of the list $(a\ b\ c)$. The empty set is always represented by NIL. The set data structure is used to represent the reduction set (R), the unprocessed portion of $R \times R$ (RR), critical pair equations (CP), and delayed critical pairs (D).

2. E-Matching with Conditional Reductions.

Because most completion and E-completion procedures spend over ninety percent of their run time applying reductions in order to conflate critical pairs, it is

essential that we have an efficient algorithm for this operation. Within the process of applying a single reduction the dominant operation with respect to time usage is the process of finding a matcher between the reduction and the term to be reduced.

The E-matching routine used in our program is largely due to Peterson [*Pe*88], who developed an efficient E-matching routine relative to empty and ACI equational theories, for reductions with conditions of the type we have defined. We have extended Peterson's routine to handle E-unification relative to C and AC equational theories. The distinguishing features of this E-matching routine are:

(1) It has the generality to handle empty, C, AC, and ACI equational theories with very little separate code for each class of equational theories.

(2) No diophantine equations are ever generated or solved. Because we only need a single matcher and not a complete set of matchers, it is not necessary to deal with this subproblem which adds a great deal of complexity and time usage to the AC and ACI unification processes.

(3) Conditions on reductions are exploited to the fullest extent possible. When a matcher is being developed, all paths which would result in a matcher which violates the conditions on the reduction are pruned as soon as they are encountered. No time is wasted developing a matcher which cannot be used because it violates the condition.

(4) Natural constraints are exploited to a large extent. For example, if there are constants or operators in the pattern which do not appear in the term to be matched, the match fails immediately. This simple test is performed before a match is partially developed, only to discover that it cannot be completed. Another example deals with the number of occurrences of variables in the pattern. If we have a pattern such as $x + x + y$ then we know immediately that either x must take on the identity or there must be some subterm in the term to be matched which occurs an even number of times. This concept generalizes for

any number of occurrences of variables in the pattern. Several other constraints such as these are used to increase the efficiency of the E-matching routine.

Since the terms we are trying to conflate contain variables, we must create the illusion that the variables in the term to be matched are actually constants, but only while the match is taking place. To do this we first rename the variables of the term to make them disjoint from the variables of the pattern. We then use a technique which we call *locking the variables*. This technique capitalizes on the use of the LISP property list to identify variables. As pointed out above, a variable is distinguished from a constant by the value of the *VARIABLE* property on the property list for the constant or variable name. We *lock* a variable by setting the *VARIABLE* property to *NIL* and setting a new property, *LOCKED*, to *T*. After the matching process has taken place we can then *unlock* the variable by setting *VARIABLE* to *T* and *LOCKED* to *NIL*.

3. Dealing with Term Symmetry.

One observation that we have made from running the completion procedure and examining the critical pairs produced is that many of the critical pairs which are generated are redundant. The source of this redundancy is *term symmetry*. Mayfield [*Ma*88] has a complete discussion of term symmetry and its effect on both unifiers and critical pairs. For our purposes it will suffice to say that two critical pairs $l_1 \overset{R \cup E}{=} r_1$ and $l_2 \overset{R \cup E}{=} r_2$ are *symmetric* to each other whenever there is a substitution σ such that σ is a variable renaming substitution and $(l_1 \ominus r_1) \overset{E}{=} (l_2 \ominus r_2)\sigma$, where $\ominus$ is a commutative operator not appearing in any of the terms l_1, r_1, l_2, or r_2. For example, the pairs

$$\text{P1: } < x + (-z) + y \overset{R \cup E}{=} y + z + x + (-z) + (-z) > \text{ and}$$

$$\text{P2: } < u + v + (-w) \overset{R \cup E}{=} v + (-w) + (-w) + u + w >$$

are symmetric since the substitution $\sigma = \{u \leftarrow x,\ v \leftarrow y,\ w \leftarrow z\}$ gives $P1 \overset{E}{=} P2\sigma$, and σ is clearly a variable renaming substitution. Mayfield shows that whenever p_1 and p_2

are symmetric critical pairs (1) p_1 conflates via R iff p_2 conflates via R, and (2) when neither pair conflates they will yield the same reduction modulo variable renaming and modulo E. Based on these two results it is clear that only one of the symmetric pairs need be processed. The other may be discarded before attempting conflation with no effect on the completion procedure. We have seen that as many as three fourths of the critical pairs generated by ACI-unification can be discarded due to symmetry. Mayfield has developed a test which determines whether or not two terms are symmetric. The cost of using the test is usually much smaller than attempting to conflate the redundant critical pairs.

4. Using Extensions.

Although we have shown in Chapter 5 that we do not need the concept of extended reductions in order to perform E-completion, we may still want to consider using them. Our implementation of the ACI-completion procedure given in Chapter 5 demonstrated that, while not essential, extensions often add to the efficiency of the AC and ACI completion processes. Once a coherence critical pair is encountered which will not conflate, a reduction may be extended and the remaining coherence critical pairs may be discarded without further processing. The extension guarantees that they will conflate. Because of the efficiency gain which this produces, we have retained the use of extensions in our E-completion program.

When a reduction needs to be extended relative to an ACI operator, we simply add the needed extension variables to the original reduction. For example, if $\lambda \rightarrow \rho$ needs to be extended for both ACI operators $+$ and $*$, we replace it with the extended reduction $u*(v + \lambda) \rightarrow u*(v + \rho)$. Clearly this one reduction provides the functionality of the three reductions $\lambda \rightarrow \rho$, $v + \lambda \rightarrow v + \rho$, and $u*\lambda \rightarrow u*\rho$ since u and v may each take on an identity and collapse away. Peterson et al. [*PB*87] showed that such an extended reduction will cohere with the associative laws for both ACI operators $+$

and $*$. When a reduction needs to be extended relative to an AC operator, we retain the original reduction intact and add one new reduction for each AC operator. This is precisely the method of Peterson-Stickel [*PS*81] and is necessary because the extension variables cannot collapse away.

If the extended reductions were left in the form just described, coherence would be guaranteed, as we have pointed out. Unfortunately, we would also like to keep the reduction set inter-reduced at all times during the completion process. We normally simplify a reduction to be added to R by the other reductions in R before making the addition, and we check after adding a reduction to see if any of the old reductions either conflate or simplify via the new R. When the extended form of a reduction has just the right form, it is possible that this simplification may move the extension variable and remove the ability of the extended reduction to provide coherence. Consider the case when we are adding a reduction such as $x*0 + x*y \rightarrow x*y$ for coherence, and the set R contains the distributive law $x*(y + z) \rightarrow x*y + x*z$. The reduction to be added needs to be extended to satisfy coherence with the associative law for $*$. However, after extension the new reduction $v*(x*0 + x*y) \rightarrow v*x*y$ simplifies via the distributive law giving $v*x*0 + v*x*y \rightarrow v*x*y$. This new reduction does *not* satisfy the coherence property with the associative law for $*$. For this reason, other researchers have enforced protection schemes which protect certain extended reductions from simplification and/or deletion during the completion process.

We have chosen to avoid treating extensions differently than other reductions and to avoid any sort of protection schemes by simply ignoring this potential problem during the completion process. This creates the possibility that our program may halt with what should be a complete set of reductions when, in fact, it is not. This situation is easy to detect, however, by running a coherence check on the final set of reductions. If all reductions satisfy the coherence property, then the reduction set is

actually complete. If they do not satisfy coherence, then it is not. In our experience the program has never found a potentially complete set of reductions which was not complete. This holds in spite of the fact that our program has encountered the situation described above where the distributive law potentially destroyed the coherence property. As long as the troublesome reduction does not make it into the final complete set of reductions this will never cause a problem.

5. User Interface.

The purpose of this section is to briefly describe the user interface for the ACI-completion program. The program is designed to be executed from within a Common LISP environment. We will describe only the LISP functions needed to define the equational theory for each operator, define the remainder of the algebraic structure, and invoke the completion procedure.

As explained earlier in the section on data structures, the equational theory for each operator is maintained on the LISP property list for the operator. The following LISP functions are designed to define the equational theory for an operator: (MAKE-C *operator*), (MAKE-AC *operator*), and (MAKE-ACI *operator identity*). For example, (MAKE-ACI '+ 0) defines + as an ACI operator with identity 0. This means that the laws $(x+y)+z=x+(y+z)$, $x+y=y+x$, and $x+0=0$ are understood to be associated with this operator and that ACI-unification and ACI-matching may be used whenever + is encountered. For AC and C theories the equational theory is specified in a similar manner, with the exception that no identity is specified. An operator which is not defined to be ACI, AC, or C is assumed to be associated with the empty equational theory, and standard unification and matching are used.

That part of the definition of the algebraic structure which is not defined in the equational theories for the operators is normally defined via critical pair equations in the variable CP which is passed to the completion procedure. Each equation is defined as a list of two items, as described above in the section on data structures. Let us suppose that we want to define the structure of an abelian group. First we define + to be an ACI operator using the MAKE-ACI function. We then define a single equation CP1 to be the additive inverse law, $x + (-x) = 0$, using the statement (SETQ CP1 '((+ x (- x)) 0)). Since this is the only additional equation needed to complete the definition of the abelian group, we next create the set CP of equations using (SETQ CP (LIST CP1)).

Whenever the definition of the structure being defined is a superset for the definition of another structure for which we already have an E-complete set of reductions, we may define the variable R as the complete set of reductions for the substructure and leave the equations which define the substructure out of CP. For example, a commutative ring with unit element is defined by the associative, commutative, and identity laws for + and *, the additive inverse law for +, and the distributive law for * over +. If we already have a complete set of reductions for an abelian group, which is a substructure of the ring, we may put the complete set of reductions for the group in R, and only the distributive law in CP. The reductions are each defined by a list of the form ($\Theta(\lambda \rightarrow \rho)$ λ ρ). For example, we might define the first reduction by (SETQ R1 '(*NIL* (+ x y (- y)) x)). When all of the reductions have been defined in this manner, the complete reduction set R is defined by (SETQ R (LIST R1 R2 ...)).

Because the data structure calls for variables to be distinguished from constants and operators by the *VARIABLE* property on the property list, each variable in CP and R must be flagged as such. To facilitate this we adopt a naming convention for variables. Our naming convention is that all atom names beginning with the letters s

through z will be considered to be variables. The ACI-completion procedure will begin by marking the property list for variables according to the naming convention and then standardizing the variable names such that they become $x1$, $x2$, , ..., xn according to their first occurrence moving from left to right across R and then CP. This is done in order to facilitate the process of keeping the variables of a term which is being reduced disjoint from the variables of the reduction set which is being applied to it.

After the sets R and CP have been defined as above, the ACI-completion procedure is invoked by the LISP function (ACI-COMPLETION R CP). If a complete set of reductions is found this function will return the reduction set. If the procedure fails it will return *NIL*. Pertinent data relating to the program's progress will be output along the way.

B. RESULTS

We now present examples which show the results of running the ACI-completion program for several different algebraic structures. The first group of examples demonstrates that the program is able to generate ACI-complete reduction sets. The second group of examples serves to demonstrate that sufficient generality has been maintained to handle some other equational theories which are subsets of ACI theories. All of the examples presented were run on an IBM PC-RT with ten megabytes of main memory, using LUCID COMMON LISP under the AIX operating system. All examples were run using the same program, with the exception that a different weighting function was used for Example 7. Except where noted, the examples were run using the weighting function ω_1 which is defined as follows:

$$\omega_1(constant) = 2 \qquad \omega_1(variable) = 2$$

$$\omega_1(x \oplus y) = \omega_1(x) + \omega_1(y) + 5 \qquad \omega_1(x + y) = \omega_1(x) + \omega_1(y) + 5$$

$$\omega_1((\ominus x)) = 2 + 2*\omega_1(x) \qquad \omega_1((-x)) = 2 + 2*\omega_1(x)$$

$\omega_1(x \circledast y) = \omega_1(x) * \omega_1(y)$ $\omega_1(x * y) = \omega_1(x) * \omega_1(y)$

$\omega_1(H(constant)) = 3$ $\omega_1(H(variable)) = 3$

$\omega_1(H(x)) = 5 + \omega_1(x)$

1. ACI-Complete Reduction Sets.

As pointed out in previous chapters, no prior E-completion theory or E-completion program has been able to generate or verify E-complete reduction sets for any equational theory which generates an infinite congruence class. The following examples of ACI-complete reduction sets illustrate that the theory presented in chapters 4 and 5 can be successfully implemented. Thus we demonstrate that the E-completion problem has been solved for a large subclass of equational theories which generate infinite congruence classes.

a. Example 1: Commutative groups. The program was given the additive inverse axiom $x + (-x) = 0$ as a critical pair equation along with the declaration that $+$ is an ACI operator with identity 0. It generated a complete set of reductions as follows:

R1: $u + v + (-v) \rightarrow u$ from input equation

R2: *If* $u \neq 0$ *then* $v + (-(w + u)) + u \rightarrow v + (-w)$ from R1 with itself

R1 deleted

R3: $(-0) \rightarrow 0$ from R1 simplified

• R4: $(-(-u)) \rightarrow u$ from R1 with itself

R5: $u + (-((-v) + w)) + (-(v + x)) \rightarrow u + (-(x + w))$ from R2 with itself

R6: $u + (-((-(v + w)) + x)) + (-v) \rightarrow u + (-x) + w$ from R2 with itself

R7: $(-((-u) + v)) \rightarrow (-v) + u$ from R2 with itself

R5, R6 deleted

• R8: *If* $u \neq 0$ *and* $v \neq 0$ *then* $(-(v + u)) \rightarrow (-u) + (-v)$ from R5 simplified

R2, R7 deleted

R9: $u + (-v) + (-w) + v \rightarrow u + (-w)$ from R2 simplified

• R10: $u + (-v) + v \rightarrow u$ from R3 with R9

R3, R9 deleted

Thus an ACI-complete set of reductions for free commutative groups consists of R4, R8, and R10. In all of our examples we will mark reductions retained in the final set with •. Note that R1 was generated as an *extended* form of the additive inverse law in order to satisfy coherence. Note also that many of the generated reductions have been generated as conditional reductions in order to maintain termination of the rewriting relation being used. Finally, we point out that R1 was removed from the set of reductions early in the completion process, but then reappeared as R10 near the end.

b. Example 2: Commutative rings with unit element. The program was given the ACI-complete set of reductions for free commutative groups as an initial R, the distributive axiom $x*(y+z) = x*y + x*z$ as a critical pair equation, and the declarations that + and * are ACI operators with identities 0 and 1, respectively. No inferences were attempted among the first three reductions. The program found a complete set of reductions as follows:

R1: $u + v + (-v) \rightarrow u$ given

• R2: $(-(-u)) \rightarrow u$ given

• R3: *If* $u \neq 0$ *and* $v \neq 0$ *then* $(-(u+v)) \rightarrow (-v) + (-u)$ given

• R4: *If* $u \neq 0$ *and* $v \neq 0$ *and* $w \neq 1$
then $(u+v)*w \rightarrow (w*u) + (w*v)$ from input equation

R5: *If* $u \neq 1$
then $v + (w*u*x) + (w*u*0) \rightarrow v + (w*u*x)$ from condition on R4

R6: $u + ((-v)*w) + ((-x)*w) + (w*v) + (w*y) +$
$(w*z) + (w*x) \rightarrow u + (w*z) + (w*y)$ from R1 with R4

R7: $u + (-(v*0)) + (-(v*w)) + (-x) + (-y) \rightarrow$

$u + (-(v*w)) + (-y) + (-x)$	from R3 with R5
R8: $u*(-v)*0 \rightarrow u*0$	from R1 with R5
• R9: *If* $u \neq 1$ *then* $u*0 \rightarrow 0$	from R1 with R5
R5, R7, R8 deleted	
R10: $u + ((-v)*w) + ((-x)*w) + (w*v) + (v*y) + (w*x) \rightarrow u + (w*y)$	from R6 with R9
R6 deleted	
R11: $u + ((-v)*w) + (w*v) + (w*x) + (w*y) \rightarrow u + (w*y) + (w*x)$	from R6 with R9
R10 deleted	
R12: $u + ((-v)*w) + (w*v) + (w*x) \rightarrow u + (w*x)$	from R10 simplified
R1, R11 deleted	
R13: $u + ((-v)*w) + (w*v) \rightarrow u$	from R9 with R12
R12 deleted	
R14: $u + (-((-v)*w)) + (-(w*v)) + (-x) + (-y) \rightarrow u + (-y) + (-x)$	from R3 with R13
R15: *If* $u \neq 1$ *then* $v*(-1)*u \rightarrow v*(-u)$	from R13 with itself
• R16: *If* $u \neq 1$ *then* $(-v)*u \rightarrow (-(u*v))$	from R13 with itself
R13, R14, R15 deleted	
• R17: $u + (-v) + v \rightarrow u$	from R13 simplified

The ACI-complete set of reductions consists of R2, R3, R4, R9, R16, and R17.

c. Example 3: Boolean rings. Following the pattern of Hsiang [*Hs*85] we examined the boolean ring defined by the axioms

A1: $x \oplus 0 = x$	A5: $x*x = x$
A2: $x \oplus y = y \oplus x$	A6: $x*1 = x$
A3: $(x \oplus y) \oplus z = x \oplus (y \oplus z)$	A7: $(x*y)*z = x*(y*z)$
A4: $x \oplus (-x) = 0$	A8: $x*(y \oplus z) = x*y \oplus x*z$

where $\oplus$ is the EXCLUSIVE OR operator, $*$ is the AND operator, 0 is false, and 1 is true. The theorems T1: $x*y = y*x$ and T2: $x \oplus x = 0$ are known consequences of the axioms of the boolean ring. Hsiang has shown that a boolean algebra, for which there is no complete set of reductions, can be imbedded in a boolean ring by rewriting the formulae of the algebra in terms of only $\oplus$ and $*$. Thus one can obtain a canonical rewriting system for boolean algebras via a canonical rewriting system for boolean rings. In order to use the ACI-completion program on the boolean ring we began with A4, A5, A8 and T2 as critical pair equations in CP, and implicitly included A1, A2, A3, A6, A7 and T1 by declaring $\oplus$ and $*$ to be ACI operators with identities 0 and 1, respectively. The program found a complete set of reductions via the sequence:

R1: *If* $u \neq 0$ *and* $v \neq 0$ *then* $(-(v \oplus u)) \rightarrow (-v) \oplus (-u)$	given
• R2: *If* $u \neq 0$ *and* $v \neq 0$ *and* $w \neq 1$ *then* $w*(v \oplus u) \rightarrow (w*v) \oplus (w*u)$	given
R3: *If* $u \neq 1$ *then* $u*(-v) \rightarrow (-(u*v))$	given
R4: $(-(-u)) \rightarrow u$	given
R5: $u \oplus (-v) \oplus v \rightarrow u$	given
• R6: *If* $u \neq 0$ *then* $u*0 \rightarrow 0$	given
• R7: *If* $u \neq 1$ *then* $v*u*u \rightarrow v*u$	from input equation
• R8: *If* $u \neq 0$ *then* $v \oplus u \oplus u \rightarrow v$	from input equation
• R9: $(-u) \rightarrow u$	from R3 with R7
R1, R3, R4, R5 deleted	

The ACI-complete set of reductions for a boolean ring consists of R2, R6, R7, R8, and R9. This set is very similar to the AC-complete set found by Hsiang.

d. Example 4: Group homomorphisms. The following gives a derivation of an ACI-complete set of reductions for a homomorphism from one abelian group into another. We began by placing in R the complete set of reductions for an abelian group over the operators $(+, 0, -)$ and the complete set of reductions for an abelian group over the operators $(\oplus, 0', \ominus)$, where $+$ and $\oplus$ are declared to be ACI operators with identities 0 and $0'$, respectively. We then placed the additional defining equation $H(x+y) = H(x) \oplus H(y)$ in CP and executed the program. No inferences were attempted among the first six reductions. A complete set was found as follows:

• R1: $u + v + (-v) \rightarrow u$	given
• R2: $(-(-u)) \rightarrow u$	given
• R3: *If* $u \neq 0$ *and* $v \neq 0$ *then* $(-(u+v)) \rightarrow (-u) + (-v)$	given
• R4: $u \oplus v \oplus (\ominus v) \rightarrow u$	given
• R5: $(\ominus(\ominus u)) \rightarrow u$	given
• R6: *If* $u \neq 0'$ *and* $v \neq 0'$ *then* $(\ominus(v \oplus u)) \rightarrow (\ominus v) \oplus (\ominus u)$	given
• R7: *If* $u \neq 0$ *and* $v \neq 0$ *then* $H(u+v) \rightarrow H(u) \oplus H(v)$	from input equation
R8: $u \oplus H(0) \oplus H(v) \rightarrow u \oplus H(v)$	from condition on R7
R9: $u \oplus (\ominus H(0)) \oplus (\ominus H(v)) \oplus (\ominus w) \oplus (\ominus x) \rightarrow u \oplus (\ominus H(v)) \oplus (\ominus x) \oplus (\ominus w)$	from R6 with R8
R10: $u \oplus (\ominus H(0)) \oplus H(v) \rightarrow u \oplus H(v)$	from R4 with R8
• R11: $H(0) \rightarrow 0'$	from R4 with R8
R8, R9, R10 deleted	
R12: $u \oplus H((-v)) \oplus H((-w)) \oplus H(w) \oplus H(x) \oplus H(y) \oplus H(v) \rightarrow u \oplus H(y) \oplus H(x)$	from R1 with R7
R13: $u \oplus H((-v)) \oplus H((-w)) \oplus H(w) \oplus H(x) \oplus H(v) \rightarrow u \oplus H(x)$	from R11 with R12
R12 deleted	
R14: $u \oplus H((-v)) \oplus H(v) \oplus H(w) \oplus H(x) \rightarrow u \oplus H(x) \oplus H(w)$	from R11 with R12
R13 deleted	

R15: $u \oplus H((-v)) \oplus H(w) \oplus H(v) \rightarrow u \oplus H(w)$ — from R13 simplified

R14 deleted

R16: $u \oplus H((-v)) \oplus H(v) \rightarrow u$ — from R11 with R15

R15 deleted

R17: $u \oplus (\ominus H((-v))) \oplus (\ominus H(v)) \oplus (\ominus w) \oplus (\ominus x) \rightarrow u \oplus (\ominus x) \oplus (\ominus w)$ — from R6 with R16

• R18: $H((-u)) \rightarrow (\ominus H(u))$ — from R4 with R16

R16, R17 deleted

The ACI-complete set of reductions for a group homomorphism is thus R1-R7, R11, and R18.

e. Example 5: Ring homomorphisms. Following the pattern of [*PS*82], we also generated a complete set of reductions for a homomorphism from one commutative ring with identity into another. The setup for this problem was very much like the previous example. We gave the program the complete set of reductions for each of the rings $(+, 0, -, *, 1)$ and $(\oplus, 0', \ominus, \circledast, 1')$ where $+$, $*$, $\oplus$ and $\circledast$ were declared to be ACI operators with identities $0, 1, 0'$, and $1'$, respectively. The additional axioms $H(x+y) = H(x) \oplus H(y)$ and $H(x*y) = H(x) \circledast H(y)$ were given as equations. No inferences were attempted among the first twelve reductions. The derivation of the complete set is as follows:

• R1: $u + (-v) + v \rightarrow u$ — given

• R2: $(-(-u)) \rightarrow u$ — given

• R3: *If* $u \neq 0$ *and* $v \neq 0$ *then* $(-(u+v)) \rightarrow (-u) + (-v)$ — given

• R4: *If* $u \neq 1$ *and* $v \neq 0$ *and* $w \neq 0$ *then* $u*(v+w) \rightarrow (u*v) + (u*w)$ — given

• R5: *If* $u \neq 1$ *then* $u*0 \rightarrow 0$ — given

• R6: *If* $u \neq 1$ *then* $u*(-v) \rightarrow (-(u*v))$ — given

• R7: $u \oplus (\ominus v) \oplus v \rightarrow u$ — given

• R8: $(\ominus(\ominus u)) \rightarrow u$	given
• R9: *If* $u \neq 0'$ *and* $v \neq 0'$ *then* $(\ominus(u \oplus v)) \rightarrow (\ominus u) \oplus (\ominus v)$	given
• R10: *If* $u \neq 1'$ *and* $v \neq 0'$ *and* $w \neq 0'$ *then* $u \circledast (v \oplus w) \rightarrow (u \circledast v) \oplus (u \circledast w)$	given
• R11: *If* $u \neq 1'$ *then* $u \circledast 0' \rightarrow 0'$	given
• R12: *If* $u \neq 1'$ *then* $u \circledast (\ominus v) \rightarrow (\ominus(u \circledast v))$	given
• R13: *If* $u \neq 1$ *and* $v \neq 1$ *then* $H(u * v) \rightarrow H(u) \circledast H(v)$	from input equation
• R14: $u \circledast H(1) \circledast H(v) \rightarrow u \circledast H(v)$	from condition on R13
• R15: *If* $u \neq 0$ *and* $v \neq 0$ *then* $H(u + v) \rightarrow H(u) \oplus H(v)$	from input equation
R16: $u \oplus H(0) \oplus H(v) \rightarrow u \oplus H(v)$	from condition on R15
R17: $u \circledast H(v) \circledast H(w) \circledast H(0) \rightarrow u \circledast H(0)$	from R5 with R13
R18: $u \circledast H((-v)) \circledast H(w) \circledast H(x) \rightarrow u \circledast H((-(x * w * v)))$	from R6 with R13
R19: $u \oplus (H(v) \circledast H(w)) \oplus H(0) \rightarrow u \oplus (H(v) \circledast H(w))$	from R13 with R16
R20: $u \oplus (H(u) \circledast H(w) \circledast H(x)) \oplus H(0) \rightarrow u \oplus (H(v) \circledast H(w) \circledast H(x))$	from R13 with R19
R21: $u \oplus (H(0) \circledast v) \oplus (H(w) \circledast v) \oplus (v \circledast x) \oplus (v \circledast y) \rightarrow u \oplus (H(w) \circledast v) \oplus (v \circledast y) \oplus (v \circledast x)$	from R10 with R16
R16 deleted	
R22: $u \oplus (H(v) \circledast H(w) \circledast x) \oplus (H(0) \circledast x) \oplus (x \circledast y) \oplus (x \circledast z) \rightarrow u \oplus (H(v) \circledast H(w) \circledast x) \oplus (x \circledast z) \oplus (x \circledast y)$	from R10 with R19
R19 deleted	
R23: $u \oplus (H(0) \circledast v) \oplus (H(w) \circledast v) \oplus (v \circledast x) \rightarrow u \oplus (H(w) \circledast v) \oplus (v \circledast x)$	from R11 with R21
R21 deleted	
R24: $u \oplus (H(0) \circledast v) \oplus (H(w) \circledast v) \rightarrow u \oplus (H(w) \circledast v)$	from R11 with R23
R23 deleted	
R25: $u \oplus (\ominus(H(0) \circledast v)) \oplus (\ominus(H(w) \circledast v)) \rightarrow u \oplus (\ominus(H(w) \circledast v))$	from R12 with R24
R26: $u \oplus (H(v) \circledast H(w) \circledast x) \oplus (H(0) \circledast x) \rightarrow u \oplus (H(v) \circledast H(w) \circledast x)$	from R13 with R24
R22 deleted	
R27: $u \oplus (H(0) \circledast H(v) \circledast w) \oplus (H(v) \circledast w) \rightarrow u \oplus (H(v) \circledast w)$	from R14 with R24

R28: $u \oplus (\ominus H(0)) \oplus H(v) \rightarrow u \oplus H(v)$ — from R7 with R24

• R29: $H(0) \rightarrow 0'$ — from R7 with R24

R17, R20, R24, R25, R26, R27, R28 deleted

R30: $u \oplus H((-v)) \oplus H((-w)) \oplus H(w) \oplus H(x) \oplus H(y) \oplus H(v) \rightarrow u \oplus H(y) \oplus H(x)$ — from R1 with R15

R31: $u \oplus H((-v)) \oplus H((-w)) \oplus H(w) \oplus H(x) \oplus H(v) \rightarrow u \oplus H(x)$ — from R29 with R30

R30 deleted

R32: $u \oplus H((-v)) \oplus H(v) \oplus H(w) \oplus H(x) \rightarrow u \oplus H(x) \oplus H(w)$ — from R29 with R30

R31 deleted

R33: $u \oplus H((-v)) \oplus H(w) \oplus H(v) \rightarrow u \oplus H(w)$ — from R31 simplified

R32 deleted

R34: $u \oplus H((-v)) \oplus H(v) \rightarrow u$ — from R29 with R33

R33 deleted

R35: $u \oplus H((-(v \circledast w))) \oplus (H(v) \circledast H(w)) \rightarrow u$ — from R13 with R34

R36: $u \oplus (H((-v)) \circledast w) \oplus (H(v) \circledast w) \oplus (w \circledast x) \oplus (w \circledast y) \rightarrow u \oplus (w \circledast y) \oplus (w \circledast x)$ — from R10 with R34

R34 deleted

R37: $u \oplus H((-(v * w * x))) \oplus (H(v) \circledast H(w) \circledast H(x)) \rightarrow u$ — from R13 with R35

R38: $u \oplus (H((-(v * w))) \circledast x) \oplus (H(v) \circledast H(w) \circledast x) \oplus (x \circledast y) \oplus (x \circledast z) \rightarrow u \oplus (x \circledast z) \oplus (x \circledast y)$ — from R10 with R35

R35 deleted

R39: $u \oplus (H((-v)) \circledast w) \oplus (H(v) \circledast w) \oplus (w \circledast x) \rightarrow u \oplus (w \circledast x)$ — from R11 with R36

R36 deleted

R40: $u \oplus (H((-v)) \circledast w) \oplus (H(v) \circledast w) \rightarrow u$ — from R11 with R39

R39 deleted

R41: $u \oplus (\ominus (H((-v)) \circledast w)) \oplus (\ominus (H(v) \circledast w)) \rightarrow u$ — from R12 with R40

R42: $u \oplus (H((-(v * w))) \circledast x) \oplus (H(v) \circledast H(w) \circledast x) \rightarrow u$ — from R13 with R40

R38 deleted

R43: $u \oplus (H((-1)) \circledast H(v) \circledast w) \oplus (H(v) \circledast w) \rightarrow u$ from R14 with R40

• R44: $H((-u)) \rightarrow (\ominus H(u))$ from R7 with R40

R18, R37, R40, R41, R42, R43 deleted

The ACI-complete set of reductions for the ring homomorphism problem consists of the 17 reductions R1-R15, R29, and R44. Note that the AC-complete set for this problem, as generated by previous E-completion procedures, contains 26 reductions.

f. Example 6: Distributive lattices. For our final example of the generation of an ACI-complete set of reductions we consider the example of a distributive lattice with identities for the lattice *meet* and *join* operators. For this example, we started the program with critical pair equations representing the absorption laws for the lattice *meet* and *join* operators, and the distributive law which distributes a *meet* across a *join*. The *meet* operator $\cap$ was declared to be an ACI operator with identity U. The *join* operator $\cup$ was declared to be an ACI operator with identity ϕ. The program found an ACI-complete set as follows:

R1: *If* $(u \neq U \text{ or } v \neq \phi)$ *and* $(u \neq \phi \text{ or } v \neq U)$
then $w \cap (u \cup v) \cap u \rightarrow w \cap u$ from input equation

R2: *If* $(u \neq \phi \text{ or } v \neq U)$ *and* $(u \neq U \text{ or } v \neq \phi)$
then $w \cup (u \cap v) \cup u \rightarrow w \cup u$ from input equation

• R3: *If* $u \neq U$ *and* $v \neq U$ *and* $w \neq 1$
then $(u \cap v) \cup w \rightarrow (u \cup w) \cap (v \cup w)$ from input equation

R2 deleted

R4: *If* $(u \neq U \text{ or } v \neq \phi \text{ or } w \neq \phi)$
and $(u \neq \phi \text{ or } v \neq U \text{ or } w \neq \phi)$
then $x \cap (u \cup w \cup w) \cap (u \cup w \cup v) \rightarrow x \cap (u \cup w)$ from R2 simplified

R1 deleted

• R5: *If* $u \neq \phi$ *then* $v \cup u \cup u \rightarrow v \cup u$ from R4 with itself

R4 deleted

• R6: *If* $(u \neq U \text{ or } v \neq \phi)$ *and* $(u \neq \phi \text{ or } v \neq U)$
then $w \cap (u \cup v) \cap u \rightarrow w \cap u$ from R4 simplified

The ACI-complete set consists of the three reductions R3, R5, and R6. As seen in previous examples, one of the reductions which is in the final set is deleted near the beginning of the process and then reappears at the end. Because of this, we have found that ACI-completion problems sometimes run more efficiently if we do not keep R simplified at all points during the completion process. We can then perform a simplification on the final result.

2. Demonstration of Generality.

As pointed out in the last section of Chapter 5, the ACI-completion procedure presented is not only able to handle ACI equational theories, but is also able to handle empty, C, and AC equational theories as well. In order to demonstrate this point we now present examples of completion relative to one member of each of these classes of equational theories.

a. Example 7: Groups using standard completion. We were able to duplicate the group example from the original Knuth-Bendix paper [*KB*70]. For this example we used the weighting function ω_2 which is defined by:

$$\omega_2(constant) = 3 \qquad \omega_2(variable) = 3$$

$$\omega_2(x*y) = (\omega_2(x) - 1)*\omega_2(y) \qquad \omega_2((x*y)^{-1}) = 2 + 5*(\omega_2(x*y) + 5)$$

$$\omega_2(x^{-1}) = 2 + 2*\omega_2(x)$$

This change of weighting function is necessary because the ω_1 weighting function gives the same weight to both sides of the associative law, which must be used as a rewrite rule in this problem. The weighting function was the only part of the program which was changed for this problem. The multiplication operator $*$ was declared to be an operator with no associated equational theory. The following three group axioms were placed in *CP*:

A1: $e*x = x$	left identity
A2: $x^{-1}*x = e$	left inverse
A3: $(x*y)*z = x*(y*z)$	associativity

The complete set of reductions modulo an empty equational theory was generated as follows:

• R1: $e*u \rightarrow u$	from input equation
• R2: $u^{-1}*u \rightarrow e$	from input equation
• R3: $(u*v)*w \rightarrow u*(v*w)$	from input equation
• R4: $u^{-1}*(u*v) \rightarrow v$	from R2 with R3
R5: $e^{-1}*u \rightarrow u$	from R1 with R4
R6: $(u*v)^{-1}*(u*(v*w)) \rightarrow w$	from R3 with R4
R7: $(e^{-1})^{-1}*u \rightarrow u$	from R4 with R5
R8: $(u^{-1})^{-1}*e \rightarrow u$	from R2 with R4
R9: $(u^{-1})^{-1}*v \rightarrow u*v$	from R3 with R8
R8 deleted	
• R10: $u*e \rightarrow u$	from R8 simplified
R7 deleted	
• R11: $e^{-1} \rightarrow e$	from R2 with R10
R5 deleted	
• R12: $(u^{-1})^{1} \rightarrow u$	from R9 with R10
R9 deleted	
• R13: $u*u^{-1} \rightarrow e$	from R2 with R12
• R14: $u*(u^{-1}*v) \rightarrow v$	from R3 with R13
R15: $u*(v*(u*v)^{-1}) \rightarrow e$	from R3 with R13
R16: $u*(v*((u*v)^{-1}*w)) \rightarrow w$	from R3 with R14
R17: $u*(v*(w*(u*(v*w))^{-1})) \rightarrow e$	from R3 with R15
R18: $u*(u*v)^{-1} \rightarrow v^{-1}$	from R4 with R15
R15 deleted	

R19: $u*(v*(w*u))^{-1} \rightarrow (v*w)^{-1}$ from R3 with R18

R17 deleted

R20: $u*((u*v)^{-1}*w) \rightarrow v^{-1}*w$ from R3 with R18

R16 deleted

- R21: $(u*v)^{-1} \rightarrow u^{-1}*v^{-1}$ from R4 with R18

R6, R18, R19, R20 deleted

The complete set of reductions consists of the ten reductions R1, R2 , R3, R4, R10-R14, and R21. This is the same result found by Knuth and Bendix [*KB*70].

b. Example 8: Latticoids using C-completion. For this example we generate a C-complete reduction set for a non-associative latticoid [*Bi*67]. This structure is a like an ordinary lattice, except that it has no associative property. It is relatively hard to come up with a structure which will yield a C-complete set of reductions. We have discovered that any structure with an associative or distributive law will cause the loss of the finite termination property in the presence of commutativity. For this example we gave the program the absorption laws for the *meet* and *join* operators, $\cap$ and $\cup$, respectively. Both operators were declared to be commutative. The C-complete set was derived via the sequence:

- R1: $(u \cup v) \cap u \rightarrow u$ from input equation
- R2: $(u \cap v) \cup u \rightarrow u$ from input equation
- R3: $u \cup u \rightarrow u$ from R1 with R2
- R4: $u \cap u \rightarrow u$ from R1 with R2
- R5: $(u \cup v) \cup u \rightarrow u \cup v$ from R1 with R2
- R6: $(u \cap v) \cap u \rightarrow u \cap v$ from R1 with R2

The C-complete reduction set consists of all six of the reductions generated. To our knowledge, this is the first time a complete set of reductions has been found for this particular structure.

c. Example 9: Commutative groups using AC-completion. As a final example we have duplicated the generation of an AC-complete set of reductions for commutative groups as is presented in [*PS*81]. We declared + to be an AC operator and defined CP to be the equations E1: $x + 0 = x$, and E2: $x + (-x) = 0$. The program formed reductions from these equations and completed the reduction set as follows:

• R1: $u + 0 \rightarrow u$	from input equation
• R2: $u + (-v) + v \rightarrow u$	from input equation
• R3: $(-u) + u \rightarrow 0$	from input equation
• R4: $(-0) \rightarrow 0$	from R1 with R3
• R5: $(-(-u)) \rightarrow u$	from R2 with itself
R6: $u + (-(v + w)) + w \rightarrow u + (-v)$	from R2 with itself
R7: $(-(u + v)) + v \rightarrow (-u)$	from R2 with itself
• R8: $(-(u + v)) \rightarrow (-u) + (-v)$	from R2 with R7
R6, R7 deleted	

The AC-complete set consists of R1-R5 and R8. This is the same result found by Peterson and Stickel. Note that R2 and R3 both came from the same input equation, E2. R3 comes from E2 directly and R2 is the extended form of E2 which is necessary for coherence.

3. ACI-Completion versus AC-Completion.

We are now able to perform a side by side comparison of both ACI-completion and AC-completion runs for the same algebraic structures. To generate the data for these comparisons we ran our E-completion procedure on the same algebraic structures for which we generated ACI-complete reduction sets in Examples 1 through 6. For each problem we changed the appropriate operators from ACI to AC and added the necessary equations to handle the identity properties as rewrite rules.

a. Step Size of Deductions.

It is interesting to compare the number of steps for a derivation of an ACI-complete reduction set to the number of steps for a derivation of an AC-complete reduction set for the same algebraic structure. Table I compares the number of inferences and related steps for our example problems. The uniformly smaller number of inferences required for ACI-completion still took us to essentially the same point as the corresponding AC-completion derivation. This clearly indicates that more distance was covered by each step, on the average. The same concept is also reflected in the smaller numbers of retained reductions. Since there are always fewer reductions in an ACI-complete set, each reduction must be able to do more, on the average, towards providing completeness. The consistently lower numbers seen in Table I reflect that the branching factor of the search space is indeed smaller when more is built into a single step. This is the result that we had hoped to see. Unfortunately, however, this does not automatically translate into a gain in efficiency. We address this issue in the following section.

Table I. COMPARISON OF NUMBER OF INFERENCES, AC VERSUS ACI

Example Problem	Inferences Made AC / ACI		Matches Attempted AC / ACI		Reductions Applied AC / ACI		Reductions Added AC / ACI		Reductions Retained AC / ACI	
1	28	14	6588	4284	231	138	8	10	6	3
2	243	31	256256	71368	1668	1735	46	17	10	6
3	39	13	9401	2517	143	41	15	9	9	5
4	105	36	27852	14310	214	77	23	18	15	9
5	555	148	443983	309378	866	1193	62	44	26	17
6	68	8	71998	20377	892	1110	17	6	11	3

b. Efficiency.

Even though an ACI-completion run takes fewer logical steps than its AC-completion counterpart, the run times shown in Table II indicate that this does not directly give an increase in efficiency. For most of the problems the AC-completion procedure runs in less time. This is especially true of the larger problems, where we had hoped that the exact opposite would happen. Examining the time spent in computing terminal forms we see that our ACI-completion procedure generally spent more time performing fewer attempted matches. We attribute this to the fact that ACI-matching is slower than AC-matching. We also see that, for the larger problems, the ACI-completion procedure spent considerably more time in the formation of critical pairs. Furthermore, only part of that difference can be attributed to ACI-unification, as reflected by the last column of Table II.

Table III gives some insight into what is going on in the formation of critical pairs. It seems that the AC-completion process generates far less redundant critical pairs. For the larger problems we often see that 75% of the critical pairs generated

Table II. COMPARISON OF TIMES, AC VERSUS ACI

Run Times (in seconds)								
Example Problem	Total Time AC / ACI		Terminal Form AC / ACI		Critical Pair AC / ACI		Unification AC / ACI	
1	53.9	56.2	25.9	38.8	17.6	8.5	12.6	4.0
2	1528	3044	1084	2017	116	737	82	316
3	51.7	40.1	23.6	27.8	6.4	2.2	2.4	1.3
4	141	144	79	67	30	43	26	19
5	2382	2872	1596	2114	116	226	78	92
6	414	1627	293	1142	68	435	35	245

by AC-completion are kept after the symmetry test. For ACI-completion, however, the program often spends a great deal of time generating a very large number of critical pairs, only to expend more time eliminating them via the symmetry test. For example, Table III shows that for the commutative ring problem of Example 2, 1381 critical pairs were computed yet only 224, or 16%, survived the symmetry test. The other 84% were redundant. This suggests that we may make a real improvement in this portion of the run time if we are able to directly generate only the asymmetric critical pairs. This issue is addressed in [*Ma*88].

Table III. COMPARISON OF NUMBER OF CRITICAL PAIRS, AC VERSUS ACI

Example Problem	Critical Pairs AC /ACI		Asymmetric Pairs AC / ACI		Percent Retained AC / ACI	
1	132	63	93	38	70	60
2	625	1381	518	224	83	16
3	79	12	59	11	75	92
4	96	202	74	38	77	19
5	410	837	309	246	75	29
6	385	891	275	214	71	24

VII. CONCLUSION

A. SUMMARY

We have demonstrated that the real problem in developing an E-completion procedure relative to an equational theory which generates infinite congruence classes is not primarily a problem of the size of the congruence classes, as had been suggested by previous researchers. We have shown that the real problem is that of establishing the finite termination of the $\overset{R/E}{\rightarrow}$ rewriting relation. We have developed a method which solves this termination problem for the class of ACI equational theories, proving its correctness and demonstrating its feasibility by way of our implemented computer program.

We have developed and implemented a theory of ACI-completion around our conditional rewriting relation, demonstrating that this approach is general enough to handle not only ACI equational theories, but equational theories addressed by earlier E-completion procedures as well. The new theory presented is in many ways simpler than previous theories, avoiding complicated reduction protection schemes during the E-completion process and potentially avoiding the use of extended reductions altogether.

As we had hoped, the new ACI-completion procedure takes fewer inferences to find a complete set of reductions than does its AC counterpart. It also retains fewer reductions in the final complete set. Contrary to our intuition, however, this produces a degradation, not a gain, in efficiency for the larger problems. It seems that we have lessened the number of inferences required to do the job, but increased the amount of work to perform a single inference, to the extent that any gains are more than lost.

The two changes that would be needed to remedy this problem are certainly not easy to achieve. First, it will take ACI-matching and ACI-unification routines which are nearly equal in efficiency to their AC counterparts. This seems unlikely since the ACI problems seem inherently to involve more work. For example, when two terms have different root AC operators, the AC-matching routine can stop immediately with no match. When two terms have different root ACI operators, however, there may still be several paths which must be followed before the ACI-matching routine finds that there is no match. Secondly, the hope for an algorithm which will directly generate only the asymmetric critical pairs is a dim one because it will require an ACI-unification algorithm which generates only asymmetric unifiers. Such an algorithm would immediately give a minimal ACI-unification algorithm in the general case of mixed operators. This last problem has been an outstanding open problem for some time.

What all of this may be telling us is that for the problem of deciding how many axioms we should build into the equational theory for E-completion systems, the old saying "If some is good, then more is better." does not necessarily hold. Perhaps we are seeing that E should contain only the troublesome axioms which cannot be placed in R because they cause either a loss of termination, like commutativity, or a loss of generality, like associativity. Given that an axiom can be placed in either E or R without creating problems, this research would indicate that it should be placed in R. An intuitive argument for this conclusion is that when an equational axiom is placed in R it is limited to use in only one direction. When it is placed in E it is undirected, and thus less constrained in its usage. It is impossible to know whether or not these conclusions are valid until other similar problems have been studied.

B. FURTHER RESEARCH

As with most research projects we have generated more questions than answers, leaving much to be addressed in the future. We consider the following to be the most interesting questions to be addressed:

(1) Can the methodology developed here to deal with one class of equational theories which generate infinite congruence classes be applied to other such classes? Can we handle equational theories containing idempotency and equipotency laws in a similar manner? We conjecture that this can be done. Better yet, can this approach be generalized further so that we may have a general E-completion procedure for equational theories, with no restrictions on the size of their generated congruence classes?

(2) Can the concept of using conditions to achieve finite termination be applied in other ways? For example, the abelian group problem might be solvable via a C-completion procedure, but the associative law leads to a loss of finite termination in the presence of the commutativity law. We believe it may be possible to attack this problem by generalizing our conditions, which are conditions on variables in reductions, to include conditions on operators in reductions.

(3) Can the theories of E-completion and completion for conditional reductions be completely blended together? We have demonstrated for a limited case that they can. It seems likely that they can in general. As part of this problem one must also address the issue of how we will mix both syntactic conditions, such as we have used for finite termination, and semantic conditions, such as we find in the reduction *If* $x \neq 0$ *then* $x*x^{-1} \rightarrow 1$.

(4) Can we improve the efficiency of the ACI-completion procedure presented? Obviously the ACI-matching and ACI-unification algorithms are good places to start. We would especially benefit from a minimal ACI-unification algorithm for

the general case of mixed operators. Better still would be an algorithm which efficiently generates only those unifiers which lead to asymmetric critical pairs, avoiding the expense of generating pairs which are also costly to discard. Another potential improvement would be the development of an ACI-unification algorithm which exploits the conditions on the reductions in much the same manner as the ACI-matching algorithm described. We also believe that it may be possible to build critical pairs which contain no extraneous variables, thus saving the cost of post-processing via identity substitution to eliminate them. Finally, we see many opportunities for the exploitation of parallelism in the E-matching, E-unification, and E-completion processes. It may be possible, for example, to develop a parallel ACI-matching routine which is as efficient as its AC counterpart.

Each of these issues should be addressed.

REFERENCES

[*Bi*67] Birkhoff, G. (1967). *Lattice Theory*, American Mathematical Society Colloquium Publications, volume 25, American Mathematical Society, Providence, RI.

[*BK*86] Bergstra, J., and Klop, J. (1986). "Conditional rewrite rules: confluence and termination." *Journal of Computer and System Sciences*, volume 32, pp. 323-362.

[*BP*87] Bachmair, L. and Plaisted, N. (1987). "Completion for rewriting modulo a congruence." Proceedings of the Conference on Rewriting Techniques and Applications, Bordeaux, France, P. Lescanne, ed., *Lecture Notes in Computer Science*, volume 256, Springer-Verlag, Berlin, pp. 192-203.

[*Bu*83] Bundy, A. (1983). "Rewrite rules." *The Computer Modelling of Mathematical Reasoning*, Academic Press, pp. 115-131.

[*Bu*85] Buchberger, B. (1985). "Basic features and development of the critical-pair/completion procedure." *Lecture Notes in Computer Science* volume 202, pp. 1-45.

[*Fa*84] Fages, F. (1984). "Associative-Commutative Unification." Proceedings of the Seventh International Conference on Automated Deduction, R. Shostak, ed., *Lecture Notes in Computer Science*, volume 170, Springer-Verlag, Berlin, pp. 194-208.

[*FG*84] Forgaard, R., and Guttag, J. V. (1984). "A term rewriting system generator with failure-resistant Knuth-Bendix." Technical Report, MIT Laboratory for Computer Science, Massachussets Institute of Technology, Cambridge, MA.

[*Hs*85] Hsiang, J. (1985). "Refutational theorem proving using term-rewriting systems." *Artificial Intelligence*, volume 25, pp. 255-300.

[*Hu*78] Huet, G. (1978). "An algorithm to generate the basis of solutions to homogeneous linear diophantine equations." *Information Processing Letters*, volume 7, pp. 144-147.

[*Hu*80] Hullot, J. (1980). "A catalogue of canonical term rewriting systems." Technical Report CSL-113, SRI International.

[*Hu*81] Huet, G. (1981). "A complete proof of correctness of the Knuth-Bendix completion algorithm." *Journal of Computers and System Science*, volume 23, pp. 11-21.

[*JK*86] Jouannaud, J.-P., and Kirchner, H. (1986). "Completion of a set of rules modulo a set of equations." *SIAM Journal of Computing*, volume 15, pp. 1155-1194.

[*KB*70] Knuth, D., and Bendix, P. (1970). "Simple word problems in universal algebras." *Computational Problems in Abstract Algebras*, J. Leech, ed., Pergamon Press, Oxford, England, pp. 263-297.

[*KR*87] Kaplan, S., and Remy, J. L. (1987). "Completion algorithms for conditional rewriting systems." Preliminary Proceedings of the Colloquium on the Resolution of Equations in Algebraic Structures. Lakeway, TX.

[*La*79] Lankford, D.S. (1979). "On proving term rewriting systems are noetherian." Technical Report, Mathematics Department, Louisiana Technical University, Ruston, LA.

[*Ma*88] Mayfield, B. (to appear 1988). "The role of term symmetry in equational unification and completion procedures." Ph.D. dissertation, University of Missouri-Rolla, Rolla, MO.

[*MK*82] Musser, D., and Kapur, D. (1982). "Rewrite rule theory and abstract data type analysis." *Lecture Notes in Computer Science*, volume 144, pp. 77-83.

[*OL*84] Overbeek, R., and Lusk, E. (1984). *The Automated Reasoning System ITP - User's Manual.* Technical Report ANL-84-27, Argonne National Laboratory.

[*PB*87] Peterson, G., Baird, T., Mayfield, B., Smith, B., and Wilkerson, R. (1987). "On testing a set of reductions for completeness modulo an equational theory." Unpublished manuscript.

[*Pe*88] Peterson, G. (1988). Private communication.

[*PS*81] Peterson, G., and Stickel, M. (1981) "Complete sets of reductions for some equational theories." *Journal of the Association for Computing Machinery*, volume 28, pp. 233-264.

[*PS*82] Peterson, G., and Stickel, M. (1982) "Complete systems of reductions using associative and/or commutative unification." Technical Note 269, SRI International.

[*Ro*65] Robinson, J.A. (1965). "A machine-oriented logic based on the resolution principle." *Journal of the Association for Computing Machinery*, volume 12, pp. 23-41.

[*Si*68] Sibert, E. (1968). "A machine-oriented logic incorporating the equality relation." *Machine Intelligence*, volume 4, Elsevier, pp. 103-131.

[*Si*79] Siekmann, J. (1979). "Matching under commutativity" in *Symbolic and Algebraic Computation*, E. Ng, ed., Springer-Verlag, Berlin, West Germany, pp. 531-545.

[*St*81] Stickel, M. (1981). "A Unification algorithm for associative-commutative functions." *Journal of the Association for Computing Machinery*, volume 28, pp. 423-434.

[*St*84] Steele, G. (1984). *Common LISP: The Language*, Digital Press.

[*TM*75] Tremblay, J., and Manohar, R. (1975). *Discrete Mathematical Structures with Applications to Computer Science*, McGraw-Hill, New York, NY.

[*Wi*87] Wilkerson, R. (1987). Private communication.

[*Wo*67] Wos, L., et al. (1967). "The concept of demodulation in theorem proving." *Journal of the Association for Computing Machinery*, volume 14, pp. 698-709.

[*Wo*88] Wos, L. (1988). *Automated Reasoning: 33 Basic Research Problems*, Prentice Hall, Englewood Cliffs, NJ.

[*WR*69] Wos, L., and Robinson, G. (1969). "Paramodulation and theorem-proving in first order logic with equality." *Machine Intelligence*, volume 4, Elsevier, pp.135-151.

[*Ye*85] Yellick, K. (1985). "Combining unification algorithms for confined regular equational theories." *Conference on Rewriting Techniques and Applications*, J. Jouannaud, ed., *Lecture Notes in Computer Science*, volume 202, Springer-Verlag, Berlin, West Germany, pp. 365-380.

VITA

Timothy Byron Baird was born on July 8, 1956 at Fort Leonard Wood, Missouri. He graduated from Rolla High School in Rolla, Missouri in May 1974.

He received his undergraduate education at Harding University in Searcy, Arkansas. While completing his Bachelor's degree he worked as a programmer for Harding's Administrative Computer Center from May 1977 through May 1979. In May 1979 he received the Bachelor of Arts degree in mathematics.

From August 1979 to May 1981 he was a graduate student and teaching assistant in the Department of Computer Science at the University of Missouri-Rolla in Rolla, Missouri. He was awarded the Master of Science degree in computer science in May 1981.

From June 1981 until May 1985 he taught computer science at Harding University, serving as an assistant professor. While teaching at Harding he also served three years as Director of Software Support for the Academic Computer Center.

Since June 1985 he has been on leave from Harding University and has been a graduate student at the University of Missouri-Rolla, pursuing the Ph. D. degree in computer science.

He has been married to Debra Elliott Baird since August 1977. They have three children: Steven, Daniel, and David.